Aalto University
School of Science
Master's Programme in Mathematics and Opeartions Research

Santtu Saijets

Modelling of preferences in multimodal routing algorithms

Master's Thesis
Espoo, April 17, 2018

Supervisor: Professor Harri Ehtamo, Aalto University
Instructor: M.Sc. (Tech) Taina Haapamäki

Aalto University
School of Science
Master's Programme in Mathematics and Operations Research

ABSTRACT OF MASTER'S THESIS

Author:	Santtu Saijets		
Title:	Modelling of preferences in multimodal routing algorithms		
Date:	April 17, 2018	**Pages:**	63
Major:	Systems and Operations Research	**Code:**	SCI3055
Supervisor:	Professor Harri Ehtamo		
Advisor:	Taina Haapamäki M.Sc. (Tech.)		

In this thesis, we study the ongoing change in the field of passenger transport. We focus on the required technological solutions and introduce an idea of a technological platform connecting all the transport service providers seamlessly to the available interfaces offering combined transportation services for the travellers. We present a reference architecture for the platform and identify that development is needed to more accurately model the travellers' preferences in the multimodal routing algorithms used in the platform.

Label constrained shortest path problem Dijkstra's (LCSPP-D) algorithm is one typically used to model the traveller's preferences in the journey planning. We propose two ways to improve the preference modelling with this algorithm. Firstly, the travellers should be clustered into similar groups so that the parameters describing the preferences could be shared within the group. This way more emphasis could be given to the optimization of the group specific parameters. Secondly, instead of returning journey plans using a single objective function, a set of journey plans should be returned where each would describe the travellers' preferences in different situations. Then, depending on temporary variables such as the weather, a travelling companion or the amount of luggage the traveller could select the plan most suitable for the specific situation.

We focus on the second improvement and build a test framework in order to evaluate the LCSPP-D algorithm more closely in our sample network. We define multiple models to describe the travellers' preferences and use these to return journey plans from the sample network. The results show that journey plans modelling the travellers' preferences can be returned and using the designed preference models for a single trip we can return multiple plans each describing different kind of preferences. However, further research is needed to study how well the algorithm can actually model the traveller's preferences and how the preference models used in the algorithm should be defined.

Keywords:	multimodal routing algorithms, preference modelling, mobility as a service
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Systeemi- ja operaatiotutkimus

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Santtu Saijets

Työn nimi:
Preferenssien mallinnus multimodaalisissa reititysalgoritmeissa

Päiväys:	17. huhtikuuta 2018	**Sivumäärä:**	63
Pääaine:	Systeemi- ja operaatiotutkimus	**Koodi:**	SCI3055
Valvoja:	Professori Harri Ehtamo		
Ohjaaja:	Diplomi-insinööri Taina Haapamäki		

Tässä tutkimuksessa tutustumme muutokseen, joka on käynnissä henkilöliikenteen alalla. Erityisesti meitä kiinnostavat tarvittavat teknologiset ratkaisut ja esittelemme ideamme teknologia-alustasta, joka yhdistäisi liikkumispalveluiden tarjoajat saumattomasti kaikkiin eri rajapintoihin, jotka tarjoavat keskitetysti liikkumispalveluita kuluttajille. Esittelemme viitearkkitehtuurin kyseiselle alustalle ja tätä kautta tunnistamme, että kehitystä tarvitaan ainakin parantamaan preferenssien mallinnusta reititysalgoritmeissa, joita alustassa käytetään.

Ehdotamme kahta parannusta tukemaan preferenssien mallinnusta olemassa olevia algoritmeja hyödyntäen. Matkustajat tulisi ensinnäkin luokitella ryhmiin preferenssiensä perusteella. Tätä kautta preferenssimallit voitaisiin jakaa ryhmän kesken ja enemmän panostusta voitaisiin käyttää ryhmäkohtaisten mallien kehittämiseen. Toiseksi sen sijaan, että reititysalgoritmit palauttaisivat yhden tavoitefunktion mukaan optimoituja reittejä, niiden tulisi palauttaa joukko erilaisia reittejä, jotka kaikki pyrkivät kuvaamaan matkustajan preferenssejä erilaisissa tilanteissa. Sitten riippuen vallitsevista muuttujista, kuten säästä, matkustusseurasta ja kantamusten määrästä, voisivat matkustajat valita tilanteeseen sopivimman reittisuunnitelman.

Tutkimme jälkimmäistä parannusehdotusta tarkemmin ja rakennamme kehikon, jonka avulla voimme testata reititysalgoritmeja testiverkostossamme. Määrittelemme useampia malleja kuvaamaan matkustajien preferenssejä ja haemme näiden avulla reittejä testiverkostostamme. Tulokset osoittavat, että preferensseihin mukautuvia reittiehdotuksia voidaan palauttaa ja muokkaamalla preferenssimalleja oikein on mahdollista palauttaa samalle reitille joukko erilaisia preferenssejä kuvaavia reittejä. Jatkotutkimusta kuitenkin tarvitaan arvioimaan, kuinka hyviä nykyiset reititysalgoritmit ovat oikeastaan kuvaamaan matkustajan preferenssejä ja kuinka ryhmäkohtaiset preferenssimallien parametrit tulisi tarkemmin määrittää.

Asiasanat:	multimodaaliset reititysalgoritmit, preferenssien mallinnus, liikkuminen palveluna
Kieli:	Englanti

Acknowledgements

First and foremost, I would like to thank Taina Haapamäki and the whole team at Flou Solutions Ltd for all the help and support you have provided for me during the process. Applying mathematics to the field of passenger transport has been an interesting journey and you have taught me a lot about both the industry and other things such as team work, communication and problem solving. I am also grateful to my supervisor professor Ehtamo for the given guidance and constructive feedback needed to make this happen.

Biggest thanks belong to my family and friends. My family has always been supportive and has been pushing me forward when needed. My friends have been giving the much needed other things to think about and also without them this would never have been possible.

Contents

1 Introduction

In the field of passenger transport it is essential for the traveller to easily find the available transport services. Until recent years the services have been scattered and there have not been any practical tools for the traveller to inquire the possible options collectively. Hence, typically e.g. taxi and public transport have been considered as separate and mutually exclusive services. However, in certain cases it could be reasonable to utilize both during a single trip.

Furthermore, in other industries such as telecommunications we have already seen a major transformation due to the digitalization (Li and Whalley 2002). Similar effects can now be seen in passenger transport as well. The established business models have increasingly been deconstructed by service providers such as Uber and Lyft (Horpedahl 2015).

We see a need for new technological solutions such as platform architectures and multimodal routing algorithms. Haapamäki and Mäkinen (2017) presented an open technology platform integrating different market players for information exchange and to matchmake travel demand and supply. The idea is to combine all the transport service providers under the same platform and to connect these through the platform to different interfaces offering transport services for the travellers. Then, the travellers can search for journey plans and handle collective bookings and payments from the same place using e.g. their mobile devices.

The presented platform utilizes multimodal routing algorithms to find the journey plans. Our view is that development is needed to improve the modelling of preferences in routing to only suggest truly relevant travelling options. The aim for this study is to find potential routing algorithms allowing this and to design methods to improve this feature in these.

In this study we first take a look at the ongoing change in the field of passenger transport. Then, we present a reference architecture for the presented technology platform. Thirdly, we conduct a literature review to study the routing algorithms available. After this we select one of the presented al-

gorithms and study how the algorithm should be modified to improve the modelling of the traveller's preferences. Lastly, we design and implement a modular test framework which allows us to test the proposed improvements.

2 Smart Mobility Options

Transport services available have stayed the same for many decades. Only recently we have seen a change in the available options. In this section we look at the ongoing change in the field of passenger transport.

2.1 Conventional Travelling Options

For long in the field of passenger transport we have had a limited amount of travelling options. Travellers have been able to utilize either private modes (e.g. car, bike, and walking), public transport modes (e.g. bus, metro, and tram) or on-demand modes (e.g. taxi). We are interested in modelling more closely how these alternatives look from the traveller's point of view. In order to do this we constructed an illustrative model to describe the traveller's selection process when selecting the mode to be exploited.

Firstly, for the model we need to identify different factors driving the decision. We assumed that the cost is the main factor. Naturally, there are also other affecting factors such as travelling time, comfort, waiting time, and the amount of rush in the vehicles. We decided to keep the cost as its own measure and combined the other factors under a measure called a perceived service level. Naturally, travellers weight the individual measures combined under the perceived service level differently but in our approximative model the actual weights are not relevant. Secondly, we need to describe how the traveller would use the described measures in the selection process. Our assumption is that the traveller first selects the sufficient service level and, then, starts to minimize the cost of the service.

Now, we can draw the previously mentioned conventional transport modes

on a scatter plot with the cost on the x axis and the perceived service level on the y axis. Again here, it is impossible to find any exact values for the described measures since they depend on many external factors. Thus, we present our approximations for these and assume that these can be used to demonstrate the phenomena we want to elaborate later in this section. The scatter is presented in figure 1.

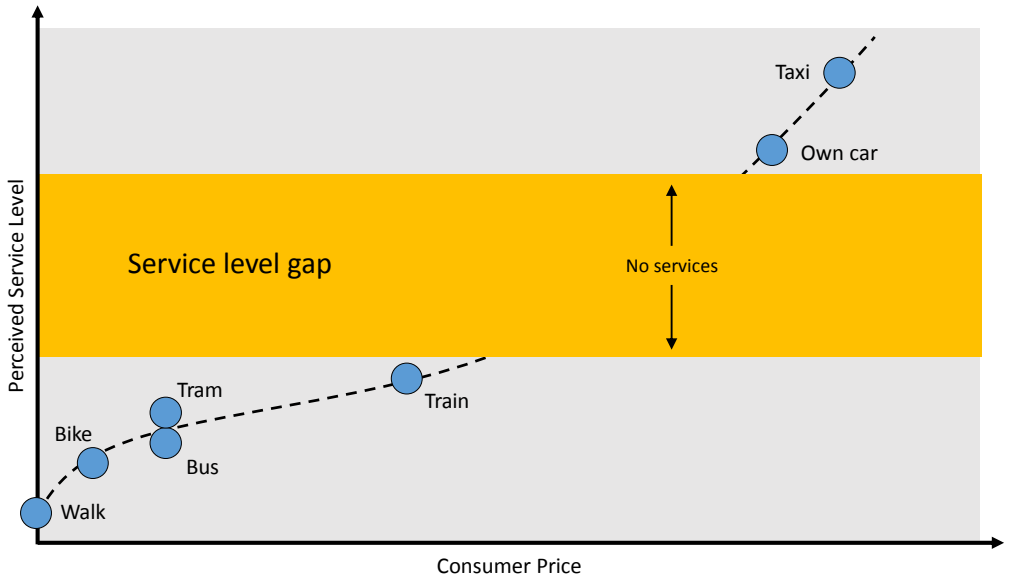


Figure 1: Estimated perceived service level and cost of the conventional travelling modes

From the figure 1 we can see that there are no services available for the whole scale of the perceived service level. The options presented in this figure seem to be divided into two clusters. In the lower part of the graph we have the options with a lower perceived service level and lower consumer price. In the upper part of the graph we have the services with a higher price and higher service level. Our approximation of the situation shows that there is a clear service level gap in the figure. Related to the decision model described earlier this means that the travellers looking for services with an average service level have to pay a lot more than they probably would be willing to or lower their requirements for the perceived service level significantly in order to find any travelling options.

2.2 Combined and New Transport Services

It would seem reasonable that there is room for new and more flexible services such as ride sharing or demand controlled public transport inside the gap presented in figure 1. Furthermore, the two clusters do not necessarily need to be considered separate and mutually exclusive. Instead we could combine services from both clusters to see the wanted variation in the cost and in the perceived service level.

Combinations would mean e.g. that on-demand modes could be used in the parts of the journey where there are no public transport options available or the frequency of these options are low. E.g. a traveller could start the journey by taking their own car from the starting point to the nearest train station. Then, continue by train to the station nearest to the destination. Finally, finish the journey by taking a taxi from the train station to the destination. In addition, if there would be multiple travellers taking the same route the taxi could be shared or new public transport options could be introduced for a specific part of the journey.

Naturally, development in technologies and solutions available would improve the situation even more. New data sources and real time connectivity could allow multiple improvement such as better optimization of vehicle capacities, dynamically optimized travelling routes and improved ride sharing possibilities.

We estimated the situation after the described transformation. Our estimation is presented in the figure 2. We can see that the service level of all the previously presented services is a bit improved. Furthermore, there are now options available for the whole scale of the perceived service level. This means that the traveller could now set the sufficient service level freely and, then, start to minimize the cost of the service without any compromises on the perceived service level.

There are multiple reasons why we estimated that the transition looks like the one seen between the figures 1 and 2. Firstly, new services and service models increase the competition in the market. Increased competition lowers

the prices and typically improves the service level of the available services. New interfaces and applications also make the services more accessible for the travellers. Secondly, resources could be utilized more efficiently. Thirdly, services could be more personalized for the traveller's preferences. More personalized services improve the quality of services especially in the unpleasant parts of the trip and this way the service levels of these journeys can be increased substantially.

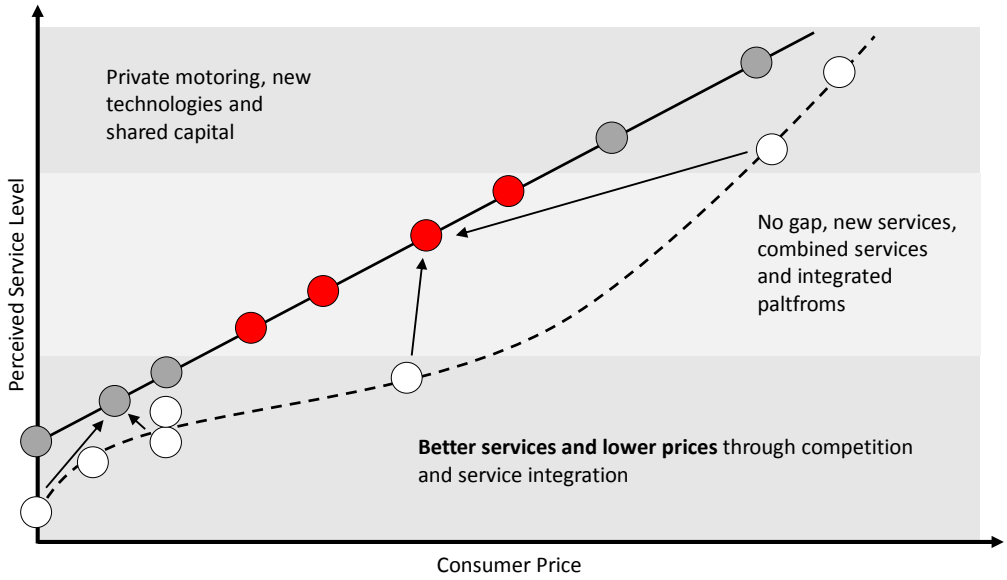


Figure 2: Our view of the situation after the combined services and new service models are fully integrated into the transport system

3 Open Technology Platform Combining Services

In the previous section we presented that there is a major change coming to the field of passenger transport. This section concentrates on a solution commonly proposed to be the driver for this change. First, we take a look at the general concept and, then, study it from a technological point of view.

3.1 Mobility as a Service Concept

Earlier we presented the idea about the combined transport services. The combination of multiple consecutive transport services is typically referred as a trip chain. As described in the previous section by trip chain we mean integration of multiple consecutive elementary connections into a single journey seamlessly. Commonly the concept where elementary connections are packed seamlessly into these trip chains is known as a Mobility as a Services (MaaS) concept.

Naturally, these trip chains will be automatically packed and easily accessible through an easy-to-use interface. Furthermore, collective bookings and payments are handled automatically. Until recent years there have not been any practical tools for the traveller to inquire the possible trip chains. In the worst case each elementary connection should have been searched, booked and paid separately. Thus, an interface is needed where all this can be done. This interface is typically known as a MaaS operator.

The MaaS operator is used through a smart phone application or a website. Given the starting point and destination it offers different kind of trip chains to reach the desired destination. The traveller can freely select the most suitable one. Then, the operator will book the needed services and deliver the payments for the necessary parties. Furthermore, the operator keeps the traveller informed about the ordered services and in case of delays or cancellations offers new options for the traveller.

The operator will be connected to a technology platform for information exchange. The platform communicates with the transport service providers and depending on the available services suggests journey plans and delivers needed bookings and payments. This platform structure is open for all relevant transport service providers and different MaaS operators are able to exploit the same technology platform structure and open API definitions.

3.2 Reference Architecture

As described above technology platforms integrating the MaaS operators and transport service providers will be needed. In this sections we present a reference architecture for this kind of solution.

The design is presented in the figure 3. On top we have the MaaS operators and in the bottom we have the transport service providers. For both of these we have the integration layers as well. In the middle we have the actual engine of the technology platform.

There are three components in the actual engine situated in the middle of the reference architecture:

- **User need and preferences** are evaluated in the service level model component situated on the right side of the reference architecture. The identification of the preferences allows to state rules as public transport can be used as long as there are no vehicle transfers included or prefer the use of bike always when possible.
- **Service supply exchange** communicates with the service providers. It knows what is the availability of different travelling options and it handles the pricing of different services. Bids can be placed by all relevant service providers and the best price will be offered for the traveller.
- **Matchmaking engine** is situated in the center of the engine and it connects the travellers to the transport service providers. It is in charge of finding the best routes and service providers included in these routes for the traveller. It is similar to the existing route planners. However, the routing algorithms take into account the user preferences from the service level model and the availability and pricing information from the service supply exchange.

Naturally, solutions described above demand development in technologies and interfaces available. This is why we also presented our estimation of the technological gap and state of the implementations available for each component presented in the infrastructure. The technological gap is color

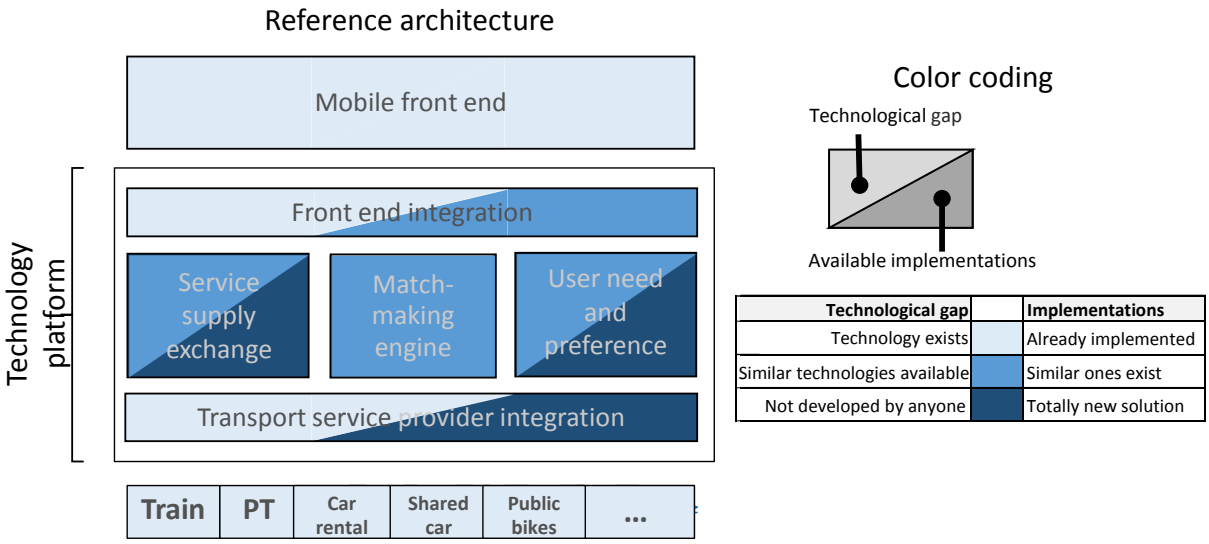


Figure 3: Our design of the MaaS-operator architecture. Technological gap and the available implementations are also color coded in the figure.

coded in the upper left corner of each component and the state of the available implementation in the right lower corner. For both attributes the estimation is done on a three scale level. The lighter the color is the more mature the attribute of the component is. The estimation is done in order to identify the parts where most research and development is needed. We can see that the largest gap and the absence of implementations is found from the operator's key components and from the interface between the operator and the service providers.

One key component in the architecture design presented above was the match making engine. In this study we focus on this component. More specifically, we take a look at the possible ways to improve the modelling of the traveller's preferences inside the routing algorithms. First, we take a look at the existing algorithms and, then, modify one to improve the routing results for our purposes.

4 Routing Algorithms

In route planning road networks are usually modelled as directed graphs because of the inherent resemblance between these two. This way it is possible to apply shortest path algorithms to find the optimal routes between two points in the road network. Shortest path algorithms have a large amount of application areas such as management science, telecommunication and transport (see e.g. Jaumard et al. 1998, Kompella et al. 1993, Dumas et al. 1991). Due to this, lots of research has been done resulting in several methods developed (Van Vliet 1978). This section presents a literature review of possible routing algorithms applied in the field of passenger transport. First, we introduce separately well performing methods for route planning in road networks and public transport networks. Then, we will combine these two resulting in a multimodal journey planner. Algorithms presented in this section are exploited later during the study.

4.1 Routing in Road Networks

Modes that are not dependent on any predetermined stops or schedules, such as car, taxi, walking, and cycling can be called individual modes. Conventional routing algorithm used in vehicle routing can be used to optimize the routes travelled by these individual modes. Naturally, different travelling speeds and possibly networks must be applied for each mode. In this section we take a look at routing algorithms that perform well in vehicle routing. First, we study how the road network can be modelled. Then, we examine shortest path algorithms and present methods that will improve the outputs of the algorithms. Finally, we take a look at preprocessing methods which help us in lowering the query time of the algorithm. In the section 4.3 we exploit these methods in the construction of the multimodal routing algorithm.

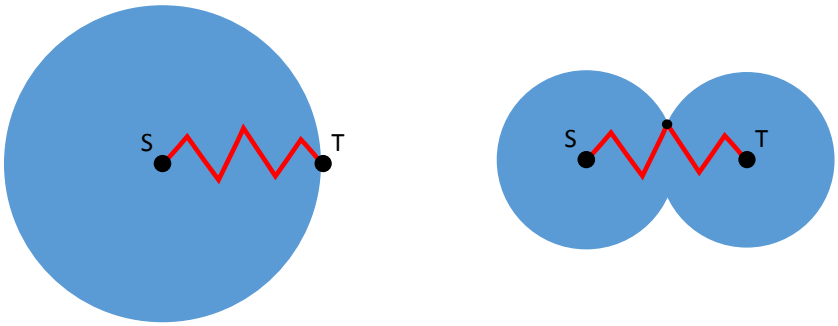
4.1.1 Modelling the Road Network

The road network is modelled as a directed graph $G = (V, A)$, where V is a set of vertices and A is the set of arcs connecting the vertices. Each intersection is represented by a vertex $s \in V$ and each road between two intersections is represented by an arc $(s, t) \in A, s \neq t$. With every arc (s, t) there is an associated non-negative cost $l(s, t)$ which corresponds e.g. to the time it takes to travel through the arc. The travelling time property of an arc is mode dependent. Distance $dist(s, t)$ between two vertices s and t is the sum of the associated arc costs in the path from s to t.

4.1.2 Dijkstra's Algorithm

In routing we are searching for the shortest path between the source vertex s and the target vertex t. Typically, Dijkstra's algorithm (Dijkstra 1959) is used to solve the shortest path problem. It computes the shortest path from a single source vertex s to every other vertex. It uses a priority queue Q ordered by distance $dist(s, u)$. Algorithm is initialized by setting all distances to infinity except $dist(s, s) = 0$ and s is added to Q. Then, algorithm iterates by extracting the vertex u with a minimum distance $dist(s, u)$ from Q. It looks at all the arcs (u, v) incident to u and for these it determines the distance $dist(s, v)$ by computing $dist(s, u) + l(u, v)$. If the value $dist(s, v)$ is improved compared to the distance saved in Q, it is updated and vertex v is added to Q. Dijkstra's algorithms has the label setting property which means that once the arc v is extracted from Q the distance $dist(s, v)$ is correct. Thus, when calculating the distance between two points s and t the algorithm may stop as soon as t has been extracted from the priority queue Q. During the computations it is typical to save the parent of each extracted vertex so that the correct route between the source s and target t can be easily found.

Dijkstra's performance can be further improved by introducing the bidirectional search (Dantzig 2016). It runs Dijkstra's algorithm simultaneously starting from the start and end points limiting the amount of ver-



(a) (b)

Figure 4: Illustrations of the search spaces that the conventional Dijkstra's algorithm and bi-directional Dijkstra's algorithm have to scan until the shortest path to the target vertex is found. Figure (a) represents the conventional Dijkstra's algorithm and figure (b) represents bi-directional Dijkstra's algorithm.

tices scanned. The algorithm can be stopped as soon as the same vertex is extracted for the first time by both algorithms. The shortest path is then

$$dist(s,t) = \min\{dist(s,u) + dist(u,t) : \forall u \text{ visited by both algorithms}\}.$$

Simultaneous running of two algorithm limits the search space of the algorithm. The amount of vertices visited is roughly the half of the amount of vertices visited with the original Dijkstra's algorithm. Figures 4(a) and 4(b) show an approximation on how the search space changes. In 4(a) we have a conventional Dijkstra's algorithm and in 4(b) a bidirectional Dijkstra's algorithm. The shape of the scanned search space depends on the way how the vertices are situated in the geographical map and, typically, it is not as symmetric as in the figures.

4.1.3 Turn Restrictions

With the algorithms above there is no way to model turning restrictions. In road networks it is typical that when approaching an intersection from a particular road it is not possible to turn to all possible directions. In the graph G this means that the possible arcs (v, w) incident to v depend on the already travelled arc (u, v). In order to find a feasible travelling route this restriction has to be taken into account in the algorithm.

One popular solution is to model these restrictions with turning tables connected to each vertex (Delling et al. 2011a). This means that the table associated with each vertex tells the allowed turning directions. There are limited amount of these tables and in order to save memory these tables can be shared among many vertices. However, if turn restrictions are applied standard Dijkstra's algorithm cannot be applied any more as shown in the figure 5. In the figure we want travel from vertex 1 to the vertex 5. Since the left turn is not allowed when travelling from 1 to 2, we can replace it with two right turns and get to the destination. Label setting property of the Dijkstra's algorithm ensures that the algorithm visits every vertex at most once and the conventional Dijkstra's algorithms would not be able to return the path to the vertex 5.

Another solution proposed by Caldwell (1961) is to present roads as vertices and connections between roads as arcs. Then, the arcs are only added between roads if the corresponding turn is allowed. This approach would allow the use of the conventional Dijkstra's algorithm. However, the size of the network increases considerably since there is one new arc for every connection in the intersection.

Third solution proposed by Geisberger and Vetter (2011) is to maintain a priority Q of arcs instead of vertices and find the optimal path between originating arc and destination arc. This would allow us to apply Dijkstra's algorithm and the size of the network would not be increased.

Turning restrictions also make it possible to apply turn costs. Applying turn costs means that the time that it takes to turn in intersection will be taking

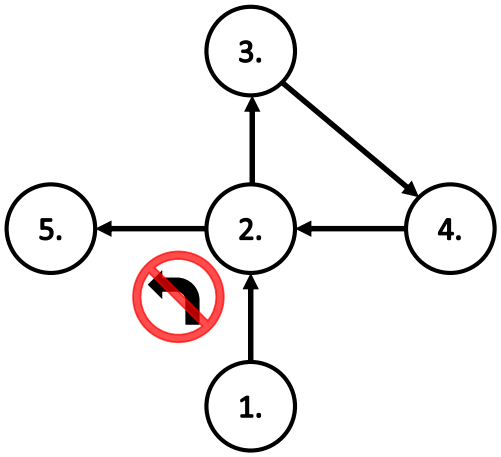


Figure 5: Example of an intersection with a turning restriction. Since the path $1 - 2 - 5$ is not allowed the conventinal Dijkstra's algorithm would not be able to return the path from 1 to 5.

into account in the routing. Turn costs can improve the outputs of the routing algorithm significantly since e.g. unnecessary U-turn are avoided. Delling et al. (2011b) propose costs of 100 seconds for U-turns and zero otherwise. Also other kinds of fixed costs can easily be applied. E.g. add a fixed cost when turning right or left and when travelling trough a intersection with traffic lights (Geisberger and Vetter 2011). Also fixed costs can be easily shared between similar intersections in the same way as turning tables in order to save memory.

4.1.4 Speed-up Techniques

For routing algorithms a fast query time is essential and it is typical that trade-offs have to be made between the preprocessing time and the query time. Dijkstra's algorithm doesn't require any preprocessing time but it also falls behind many algorithms when comparing the query times. Lots of research have been made in order to reduce the query time of the routing

algorithms (see e.g. Goldberg and Werneck 2005, Van Vliet 1978, Geisberger et al. 2012a).

Typically, in designing of these speed-up techniques three different properties of the road networks can be exploited: goal direction, hierarchical structure or small separators (Bast et al. 2016). Goal direction techniques exploit e.g. geometric properties of the network in order to guide the search algorithm into the right direction. Thus, reducing the number of unnecessary vertices scanned. Hierarchical structure techniques exploit the strong hierarchical structure of the road networks in the way that it take into account that long paths typically converge to a small number of important roads such as highways. By scanning these roads first it is possible to reduce the number of unnecessary roads scanned. Small separator techniques exploit the fact that road networks are close to planar networks. Planar networks are networks that can be drawn on to a planar surface without any arcs crossing each others. Although, road networks are not planar since there are bridges and underpasses they have been observed to have small separators as well (see e.g. Eppstein and Goodrich 2008). This means that by removing a comparatively small number of vertices the graph can be composed to several smaller graphs. Then, e.g. distances between these separating vertices can be precomputed and this information can be used in order to decrease the query time.

In this study we will present more closely a popular speed up technique called contraction hierarchies (CH) presented by Geisberger et al. (2008). This algorithm exploits the second presented property of the road networks, hierarchical structure.

CH algorithm is run in two phases. There is a preprocessing phase and a query phase. In the preprocessing phase a procedure called vertex contraction is performed. This procedure adds several short cuts into the graph. Then, in query phase bidirectional Dijkstra's algorithm uses these short cuts and is able to return the shortest path faster than the conventional bidirectional Dijkstra's algorithm. Geisberger and Vetter (2011) showed also that this technique works with turning restrictions and costs as well.

The vertex contraction is performed to every vertex in the graph one at a

time. The order of the contraction is arbitrary but in order to achieve the needed speed-up in query time, a sophisticated order is essential. Here, a single vertex contraction is presented first formally. Then, we go through a simple example of the same procedure. Finally, an approach to select the contraction order more sophisticated is presented.

Let v be the first vertex to be contracted. Furthermore, let $\{u_1, \dots u_k\}$ be the set of vertices where there exists an outgoing arc $(u_i, v), i = 1, \dots, k$ and $\{w_1, \dots w_l\}$ be the set of vertices where there exists an incoming arc $(v, w_j), i = 1, \dots, j$. First, the algorithm removes all incoming and outgoing arcs of v and checks if any shortest path from any vertex in $\{u_1, \dots u_k\}$ to any vertex in $\{w_1, \dots w_l\}$ were removed. If there were shortest paths removed, the algorithm adds short cuts to the graph corresponding to these removed shortest paths. Then the algorithm selects the next vertex to be contracted. Again it removes the incoming and outgoing arcs, checks if there were any shortest paths removed and adds corresponding short cuts to the network. The algorithm continues this until the whole graph is contracted. After each vertex contraction the graph is left as it is. Thus, no removed arcs are placed back in the graph and no added short cuts are removed between the vertex contraction. After each vertex is contracted a result graph $G*$ can be build. This is the original graph G extended with the short cuts calculated in the vertex contraction phase.

An example of a single vertex contraction procedure is presented in figure 6. In the figure we have a small graph with 6 vertices. Arc costs are also presented in the figure. In the figure the vertex 3 is contracted. Thus, first all incoming and outgoing arcs are removed from 3. Then, we can see that the shortest paths from 1 to 6 and 1 to 5 were removed. Thus, we have to include two short cuts. These are marked with red arcs. It could be possible to add also a short cut from 1 to 4 but since there are also other paths with the same cost it is not compulsory. If the whole vertex contraction phase would be performed to the example graph we would have to continue the vertex contraction from the resulted graph and perform the next contraction to a next vertex in the graph. Then, we would continue this until every vertex in the graph is contracted.

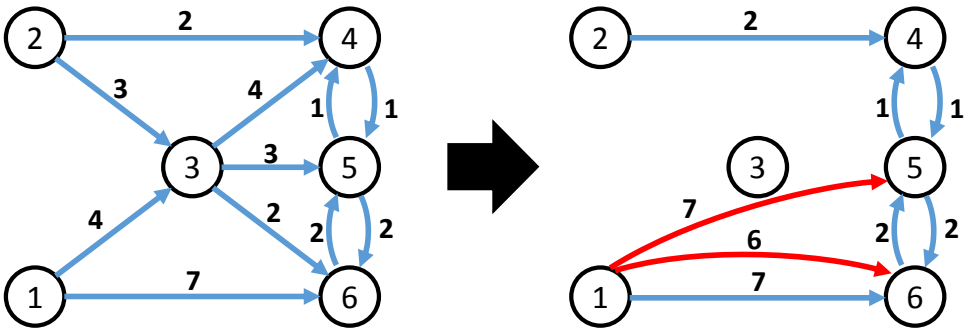


Figure 6: Illustration of contraction procedure in contraction hierarchies algorithm. In the figure vertex 3 is contracted.

As mentioned before, the order in which the vertices are contracted is arbitrary. However, it has to be done in a sophisticated way in order to achieve the needed speed-up in query time. Ideally, we would like to select the order so that only the important short cuts would be added to the graph. Thus, short cuts representing important roads such as highway and main roads. Multiple ways have been presented to select the order (Geisberger et al. 2008). The one presented most frequently in the literature is the lazy updates technique. Next, we will present this briefly.

In the lazy updates a measure called edge difference is first calculated for every vertex. It is a property of a vertex which tells that if the vertex contraction is performed to it what is the difference in the amount of arcs in the graph before and after the contraction. E.g. the edge difference of the vertex 3 in figure 6 is -3 since five arcs are removed and two short cuts are added. After the edge difference is calculated for each vertex they are placed in a priority queue ordered by the edge differences and the contraction is started from the vertex with a smallest edge difference. Naturally, when vertices are contracted, the edge difference of the non-contracted vertices can change and in principal after each contraction these values should be updated for the whole priority queue. However, the calculation of the edge difference for every vertex is considered computationally costly. This is why on every iteration the edge difference of the vertex next in the priority queue is only calculated. If it is equal or lower than the original edge difference, the vertex

will be contracted. Otherwise, it is placed back in the priority queue. Again, this procedure is repeated until the whole graph is contracted.

There is still one thing that needs to be considered while contracting vertices. In order to be able to exploit bi-directional Dijkstra's after the vertex contraction phase we have to know the order on which the contraction is performed. This is why while the vertices are contracted labels from 1 to $|V|$ are assigned to them.

Before the query phase of the CH algorithm can be applied two alternative graphs has to build from $G*$. This is done using the labels assigned to the vertices in the contraction phase. These two graphs are called up and downward graphs. Upward graph $G* \uparrow$ is build in the way that it only contains the arcs where the arc is going from a vertex with a smaller label to a vertex with a larger label. The downward graph $G* \downarrow$ is build in opposite way so that it contains only the arcs where the arc is going from a vertex with a larger label to a vertex with a smaller label. Furthermore, the arcs are reversed in $G* \downarrow$.

Next, bi-directional Dijkstra's can be applied. It is done so that the algorithm that starts from the source vertex uses the upward graph $G* \uparrow$ and the algorithm that starts from the target uses the downward graph $G* \downarrow$. The shortest path is then

$$dist(s,t) = \min\{dist(s,u) + dist(u,t) : \quad \forall u \text{ visited by both Dijkstra's algorithms}\}. \tag{1}$$

4.2 Routing in Public Transport Networks

In the previous section we examined how routing of individual modes can be done in a road network. This section examines the other popular application of routing algorithms, routing in public transport networks. Thus, here we consider only modes such as bus, metro, tram and train that depend on some predefined schedule. A key difference compared to the routing in road networks is that routing in public transport networks is inherently

time-dependent. This means that the network is given by the schedule and consists of stops and scheduled connections between them. Journeys performed in this network can consist of one or more consecutive elementary connections. These elementary connections are journeys travelled in a single vehicle between two specific stops. Footpaths have to be also included in the model in order to allow walking between two nearby stops. This section presents algorithms that allow routing in public transport networks. First, we take a look at possible ways to model the public transport network. Then, we study the algorithms applied in these networks and, finally, we examine the possible ways to fasten the query time.

4.2.1 Modelling the public transport Network

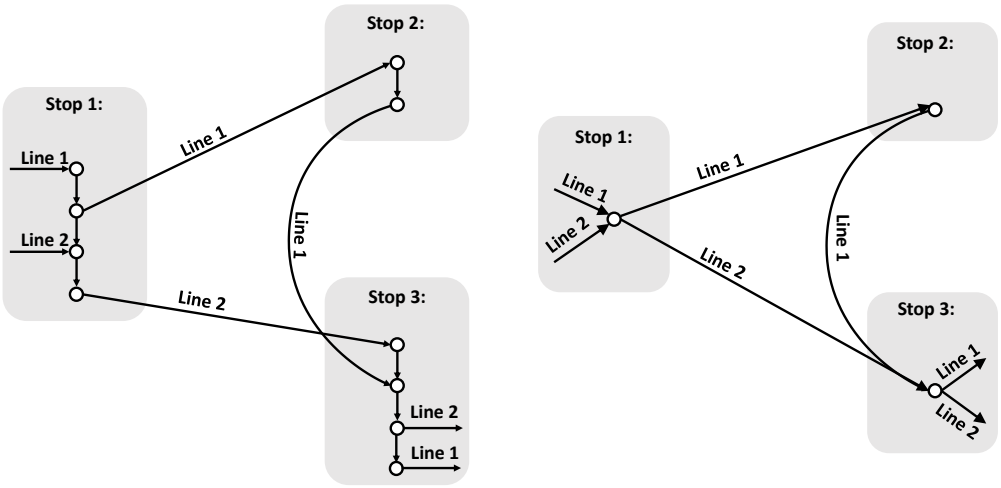
The timetable can be modelled as a directed graph $G = (V, A)$ and there are two main approaches to do this, time-expanded and time-depended model (Müller-Hannemann et al. 2007). Both of these have their advantages and disadvantages. In this section we will introduce both of these approaches side-by-side in order to be able to compare the two available methods.

First, we have the time-expanded model. Here, every vertex $s \in V$ corresponds to an event happening in a specific time. They are either vehicles arriving to a specific stop or vehicles departing from a specific stop. Thus, there exists a vertex for every event in the timetable. Naturally, there are arcs in the graph that represent transitions between the events in the timetable. An arc $(s, t) \in A, s \neq t$ between the vertices s and t will added to the graph if one of the following condition is satisfied.

- Vertices s and t represent consecutive events of a vehicle travelling between two stops. I.e., s is the departure event from a specific stop and t is the arrival event to the following stop.
- Vertices s and t represent consecutive events within a single stop. Events s and t can be e.g. the arrival and departure event of the same vehicle or two consecutive arrival events of different vehicles to the same stop. The key is that the events within a specific stop have

to be ordered by time and two consecutive events have to be connected with an arc corresponding to the possibility to just stay on the stop and wait for the upcoming departures. These arcs within a stop can be called as waiting arcs.

For every arc $(s,t) \in A, s \neq t$ there is a non-negative cost $l(s,t)$ assigned which corresponds to the time difference between the two events in the vertices s and t. Figure 7(a) shows an example of a time-expanded graph. In the figure we have a sub-graph of a larger public transport network. In the graph we have two public transport lines and three stops. The first line travels the route $stop1 - stop2 - stop3$ and the second line travels straight from $stop1$ to $stop3$. We can see that the vertices can be seen to unroll the time since there is always a single vertex for every event in time.



(a) (b)

Figure 7: Two illustrations of the same sub-graph belonging to a larger public transport network. Figure (a) represents the time-expanded model and (b) represents the time-dependent model.

Second possibility is to use the time-depended model presented by Brodal and Jacob (2004). This model is closer to the one presented in the section Routing in Road Networks. In this model vertices $s \in V$ corresponds to the stops and there is an arc $(s,t) \in A, s \neq t$ between the stops s and t if there is a vehicle going from s to t without stopping anywhere. The difference

to the model in the road networks is that the arc can be travelled only at certain times and, thus, the travelling time depends on the time of arrival to the source vertex. This information is encoded in the travel time function associated with the arc (s, t). The function outputs the complete travelling time given the arrival time in vertex s. E.g. if the traveller arrives to the departure vertex 10 minutes before the desired connection departures the travelling time is naturally the actual time it takes to travel the distance plus the 10 minutes the traveller has to wait before the departure. An example of a travelling time function is presented in figure 8. Here, we have a simple connection between two stops. The travelling time is 15 minutes and the connection departures every 20 minutes. We can see that the minimum travelling time is 15 minutes if the traveller arrives to the departure stop just in time. Furthermore, the maximum is just less than 35 minutes which corresponds to the situation where the traveller just misses a connection.

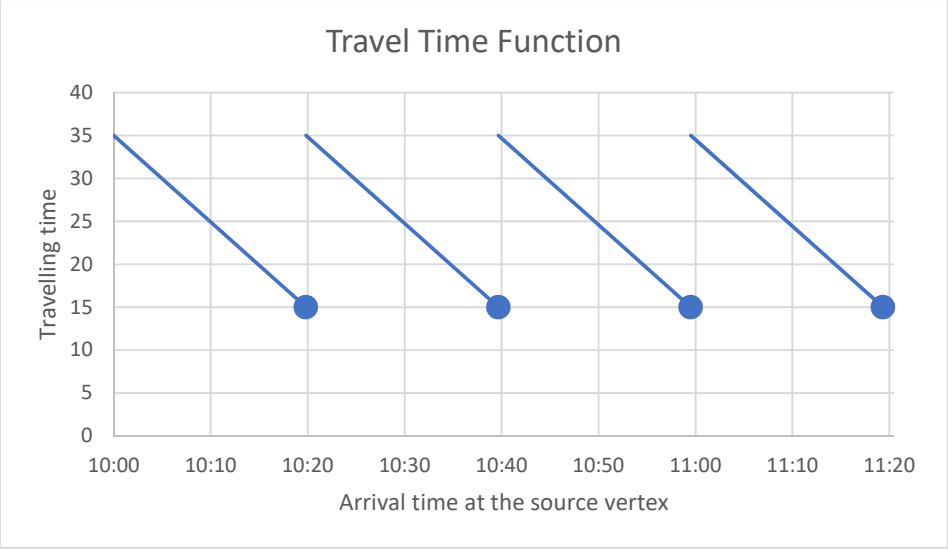


Figure 8: Example of a travelling time function. On the x-axis we have the traveller's arrival time to the source vertex and on the y-axis we have the travelling time. Filled circles represent the departure events from the corresponding stop.

Network itself in the time-depended model looks very similar to the network

used in road networks. The figure 7(b) presents the same network as in 7(a) but modelled with the time-depended model. Now there is a travelling time function associated with each of the arcs in the network.

4.2.2 Transfer Buffers

There is also a need to model the time that it takes to change the vehicle at a stop with multiple connections departing from it. There are multiple reasons to include this property in the model. Typically, the traveller needs some time to get out from the previous vehicle and some time to find the next vehicle. The previous connection can also even be a bit late and, thus, some buffer is needed. A common practice is to include a constant time into the model that the traveller at least have to have in order to be in time in the next vehicle. This constant time is called as a transfer buffer. The implementation of the transfer buffers depends on the model used to model the public transport network.

For the time-expanded model Pyrga et al. (2008) presented a solution where for each of the departure vertices an additional transfer vertex is presented. This means that all events in a specific stop can be separated into three layers: arrival, transfer, and departure layers. An example is presented next in order to demonstrate the proposed method.

A simple example of the time-expanded model with transfer buffers is presented in the figure 9. The figure presents a single stop when time is running from 10:00 to 10:12. The time is presented on the left. Between this time there are three lines arriving and departing the stop: *Line* 1, *Line* 2 and *Line* 3. Gray areas in the figure present the three layers. As seen from the figure there are always an additional transfer vertex presented next to each departure vertices. Since the time running on the left, we can see that the arc length between the transfer and departure vertices is always zero. Next, two arcs are added for each of the arrival vertices. One goes from the arrival vertex to the departure vertex of the same vehicle. It is always possible to stay in the vehicle and continue using the connection. The second arc goes

to the next transfer vertex which is at least length of the transfer buffer away from the arrival vertex. In this figure this means that if the next connection leaves in less then 5 minutes from the arrival event the corresponding transfer vertex has to be skipped. Because of this the second arc leaving from the arrival vertex of the *Line* 1 skips the transfer vertex of *Line* 2. The departure of *Line* 2 is 4 minutes after the arrival of the *Line* 1. The transfer buffer is 5 minutes and, thus, the traveller cannot make to the departure of the *Line* 2. However, the *Line* 3 leaves 10 minutes after the arrival of the *Line* 1 and 8 minutes after the arrival of the *Line* 2. Thus, departure of the *Line* 3 is the first transfer where travellers transferring from both lines *Line* 1 and *Line* 2 can make in time according to the model. This is why there are two arcs going to the transfer vertex of the *Line* 3.

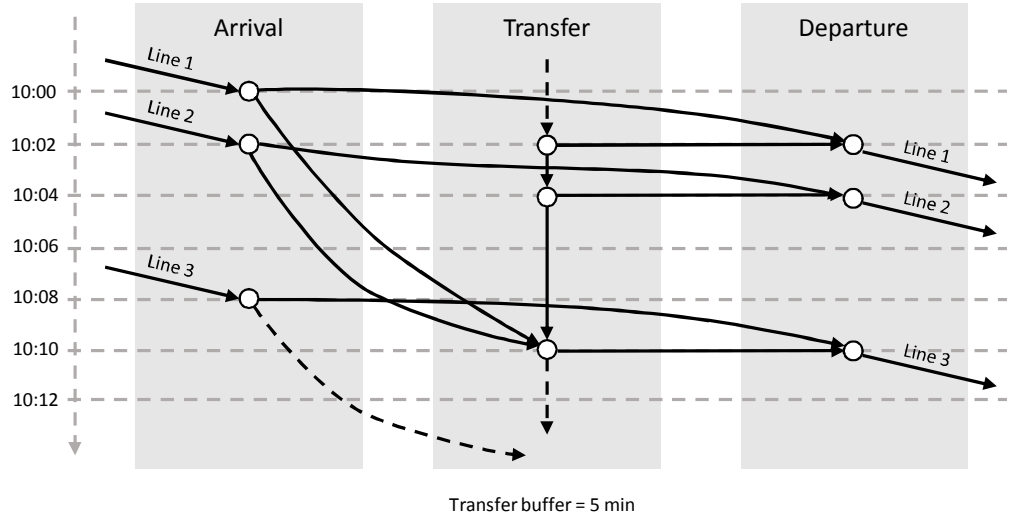
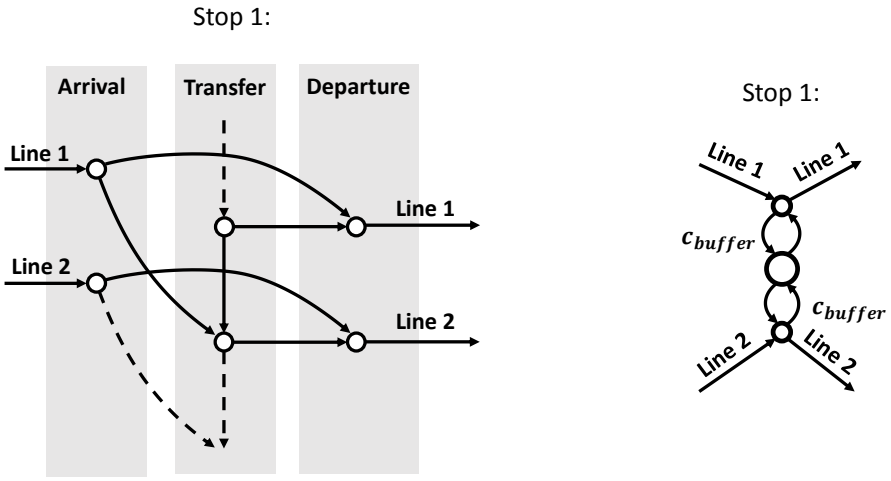


Figure 9: Example of transfer buffers applied to the time-expanded model. Here, we have a single stop with three lines arriving and departing the stop: *Line* 1, *Line* 2 and *Line* 3. Time is presented on the left side of the figure and all events happen between 10:00 and 10:12. The dashed arcs point to events outside this window.

Earlier a sample network constructed with the time-expanded model was presented in the figure 7(a). The figure 10(a) presents the stop 1 presented originally in 7(a) extended with the transfer buffer model presented above. The time is left out of this figure in order to keep the figure as simple as possible.



(a) (b)

Figure 10: Two illustrations of the same stop in a public transport network. In figure (a) the stop is modelled using the time-expanded model and in figure (b) using the time-dependent. Transfer buffers are included in the both models.

With the time-dependent model there are several possibilities on how to model the transfer buffers. Pyrga et al. (2008) presented a model where with every arrival to a specific stop there is a connection specific vertex presented. These vertices are connected to the common stop vertex with an arc that has a constant cost that correspond to the applied transfer buffer. An example of the situation is presented in 10(b). This example corresponds to the stop 1 presented in 7(b). Let's assume that that the traveller arrives to the stop using the connection $Line1$. Based on the model it first arriver to the connection specific vertex presented on top of the figure. If the traveller selects to continue using the same connection no extra cost is applied and the journey is continued normally using the travel time function associated with the following arc. On the other hand, if the traveller selects to get out from the vehicle and move to the common stop vertex presented in the middle a cost equal to the c_{buffer} has to be accepted. Then, the traveller can select without any extra costs which is the next connection to be exploited.

4.2.3 Footpaths

Naturally, also the possibility to walk between nearby stops has to be included in the model. Also here, the method on how the footpaths are applied depends on the network model applied.

In the time-expanded model the procedure is quite simple but requires addition of multiple arcs to the graph. If there exists a footpath between two stops, an arc is added between each of the arrival event at the first stop and the earliest reachable departure event at the transfer vertex in the latter stop.

E.g. Disser et al. (2008) presented a way to include footpaths in the time-dependent model. The way to model the transfer buffers in the time-dependent case makes the model a bit more complicated. Since the footpaths already include the total cost of walking between the two stops, two common stop vertices cannot just be connected with a footpath. This would result as an extra transfer buffer cost applied for each footpath exploited in the middle part of a journey. Furthermore, the transfer buffer cost cannot not be just subtracted from the added footpaths since when a journey would start with walking the cost of the footpath would be too small. This is why an additional foot vertex has to be introduced for every stop with footpaths connected to it. An example of the situation is presented in figure 11. Here, the foot vertex is presented on the left, the common stop vertex is in the middle, and footpaths are marked with dashed arrows. We can see that all departing footpaths leave from the foot vertex but arriving arcs arrive straight to the common stop vertex. This is done to avoid zero length loops that would allow avoiding of the transfer buffer cost.

For the both models possible footpaths between two stops can be calculated beforehand using the algorithms presented in the section 4.1 and then included in the public transport networks. The way of selecting the added footpaths has to be designed properly.

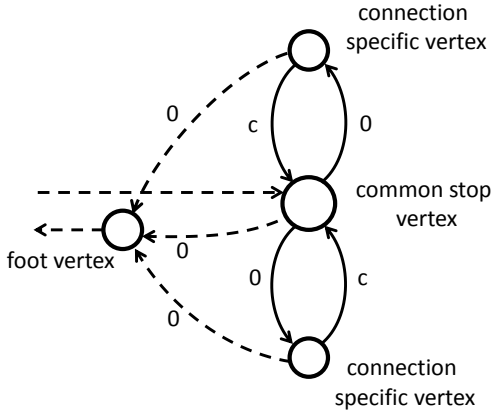


Figure 11: Example of including footpaths (dashed arrows) in the time-dependent model.

4.2.4 Applying Dijkstra's to public transport Networks

In public transport routing the network is time-dependent and the costs are modelled in time it takes to travel between the vertices. Thus, we want to minimize the time it takes to travel the route between the source an the target stop. This problem is called as the earliest arrival problem.

In the time-expanded model the graph is a normal directed graph with non-negative arc costs. Thus, it is possible to apply the same methods as presented in the road network routing. The only difference is that now there is no single target vertex. This is why the algorithm is initialized with the earliest possible event in the source stop and stopped when any vertex from the target stop is extracted from the priority queue. Algorithm can be called as Time Expanded Dijkstra (TED).

In the time-dependent model the arc costs are not constant and depend on the departure time. Thus, modifications to the conventional Dijkstra's algorithm has to be made (Orda and Rom 1990). First, instead of keeping the travelling times in the priority queue Q, the times of day are saved in the priority queue. The other modification that has to be made is that when the algorithm scans an arc (s, t) the cost $l(s, t) = c_{(s,t)}(\tau)$, where τ is the time extracted from the Q and $c_{(s,t)}(x)$ is the travel time function associated

with the arc (s, t). The rest of the algorithm works in the same way as in the previous sections and the algorithm continues until the target vertex is extracted from Q. Also here, multiple target vertices have to applied since the model to handle transfer buffers increases the number of vertices associated with a specific stop.

4.2.5 Speed-up Techniques

Similarly as in the vehicle routing, the query time is important also in public transport routing. For the both models, time-expanded and time-dependent, many speed-up techniques have been introduced (Bast et al. 2016). For the time-expanded model e.g. Bauer et al. (2008) combined construction hierarchies and goal-directed methods in order to fasten the query time. For the time-dependent model e.g. Geisberger (2010) applied contraction hierarchies but had some problems due to too many shortcuts.

4.3 Multimodal Routing

In multimodal routing we would like to combine all the travelling modes presented in the previous sections under the same algorithm. Thus, the algorithm should be able to return journeys that can exploit e.g. car, public transport modes and on-demand modes during the same trip. An example of a multimodal journey could be that the traveller starts the journey by taking a car to the nearest train station, continues by train, and finishes the journey with a taxi to the destination. In this section we first study how the multimodal network can be modelled. Then, we take a look at an example algorithm applied in multimodal routing. Finally, we consider possible speed-up techniques for the algorithms.

4.3.1 Modelling the Multimodal Network

A common way to model a multimodal network is to first build separate graphs for each of the modes as presented in the previous sections. Then,

include link arcs that allow modal switches between the separate graphs (Delling et al. 2009). These link arcs are only introduced between the vertices we want to allow the modal switches. Typically these can be presented in places like train stations and they are connected to the geographically nearest vertices in other graphs. E.g. a link arc is placed between a train station vertex and the nearest vertex in the road map. It can be also taken into account that modal switches from a private car to other modes take place only in places with park-and-ride spaces.

In order to connect two nearest vertices from separate graphs the coordinates of the vertices have to be available and so called nearest neighbour problem has to be solved. This means that we have to calculate the distance from e.g. the trains stop vertex to the nearby vertices in the road network.

A popular solution is to use k-dimensional trees (Bentley 1975). In k-dimensional trees the algorithm first splits the search space iteratively into small areas which correspond to the tree leafs. This is done so that the leaf size is relatively small and there are at least one vertex in each leaf. Then, these leafs can be exploited to calculate the distances between the nearby vertices and based on the distances the vertices to be connected with a link arc can be determined.

Since independent modes are modelled in similar graphs and the graphs share many common vertices, it seems unnecessary to apply the link arcs between them. One possible solution is to assign two separate costs to the arcs between the two vertices where several modes can be exploited (Pajor 2009). However, this would require some minor changes to the Dijkstra's algorithm.

Furthermore, since in the multimodal network walking can be performed in a network specifically intended for walking there is no need to include separate footpaths. Link arcs between the public transport network and pedestrian network need to be linked properly using the methods presented in Foothpaths section.

In multimodal routing it needs to be considered that although a modal change would be possible in a specific place it might not be feasible. E.g., it shouldn't

be possible to always switch to the car mode during a trip since obviously the car is not available everywhere. The algorithm presented next gives answers to this question.

4.3.2 Label Constrained Shortest Path Problem

Multiple algorithms have been developed for multimodal routing. In this study we will concentrate on the one called label constrained shortest path problem Dijkstra (LCSPP-D) presented by Barrett et al. (2000). As mentioned above the algorithm would have to ensure that the sequence of exploited travelling modes is feasible. LCSPP-D uses formal languages to do this. This will be demonstrated below but first some other concepts are needed to be defined. This section mainly follows the concepts presented by Pajor (2009).

First, in LCSPP-D an alphabet Σ has to be defined. It is a finite set of symbols. An example of an alphabet could be

$$\Sigma = \{foot, car, bus, train, tram, metro, bike, taxi\}.$$

After the alphabet is defined each arc in the graph G is assigned with a label from Σ based on the mode that is exploited when using the arc.

A sequence $w = [foot, car, foot]$ of symbols from Σ is called a word and a concatenation of words $w = [foot, car, foot]$ and $v = [foot, bus, bus, foot]$ is simply defined as $wv = [foot, car, foot, foot, bus, bus, foot]$. Then, language L has to be defined to restrict the sequences of used travelling modes, i.e. the accepted words.

For the language L the i'th power set of L can be defined. It is done recursively so that

- If $i = 0 :\ L^0$ is an empty word and
- If $i > 0 :\ L^i = \{wv | w \in L^{i-1}$ and $v \in L\}$.

Then, we can introduce the Kleene-Closure of L which is

$$L^* = \bigcup_{i \geq 0} L^i.$$

This allow us to define that $L \subseteq \Sigma^*$. Furthermore, we have to define that concatenation of two languages $L_1, L_2 \subseteq \Sigma^*$ is defined

$$L_1 \cdot L_2 = \{vw | v \in L_1 \text{ and } w \in L_2\}.$$

Generally, LCSPP-D puts no restriction on the language L but, typically, regular languages are considered sufficient to model the mode restrictions (Pajor 2009). Thus, also here the language L is restricted to be a regular language.

Definition 1. *Regular languages over an alphabet Σ can be defined recursively using construction rules*

1. *The empty language $\emptyset$ is regular*
2. *$\{\sigma\}$ is an regular language for all $\sigma \in \Sigma$*
3. *If L_1 and L_2 are regular languages, then $L_1 \cup L_2$, $L_1 \cdot L_2$ and L_1^* are also regular languages*
4. *There are no other regular languages over Σ*

Regular languages can be identified using regular expressions or finite automata. Next, finite automata are introduced for this purpose.

A (non-deterministic) finite automaton $A = (Q, \Sigma, \delta, S, F)$ consists of the possible states Q, an alphabet Σ, a state transition function δ, a set of initial states S and a set of final states F. Typically, finite automata are described visually with a transition graph. An example of a transition graph is presented in figure 12. States $q \in Q$ are presented as vertices and for every label $\sigma \in \Sigma$ we draw an arc from q to r if and only if $r \in \delta(q, \sigma)$. Initial states are marked with an incoming arc and final states are marked with a double framed vertex.

The transition graph is used to determine whether a word w is accepted by the language L. The word w is accepted by the finite automaton if there

exists a path from one of the initial states to one of the final states so that the subsequent arcs on the path are labelled the subsequent symbols of w. If there is no such path the word w is rejected. E.g. in the figure 12 word $v = [a, b, b, a]$ would be accepted since we can travel the path $q_0 - q_1 - q_2 - -q_3 - q_3$ from the initial state to the final state. On the other hand word $u = [a, b, a]$ would be rejected.

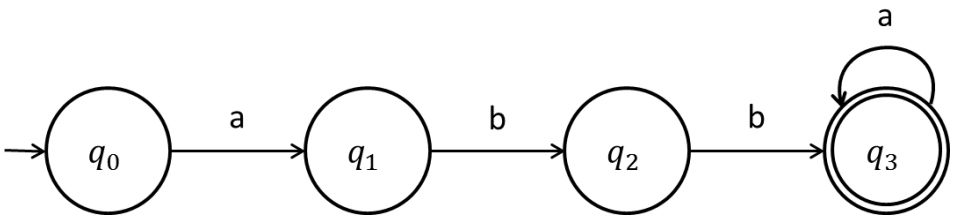


Figure 12: Example of a transition graph buil for alphabet $\Sigma = \{a, b\}$

LCSPP-D operates on a product network which is constructed using the original network and the finite state automaton. Here, we give a formal definition for the product network.

Definition 2. *Giving the original graph $G = (V, A)$, the labels from Σ for each arc and the non-deterministic finite state automaton $A = (Q, \Sigma, \delta, S, F)$, the product network $G^X = (V^X, A^X)$ is defined as follows*

- *The vertices in the product network consist of tuples $(v, q) \in V^X$ where v is a vertex from the original graph and q is a state from the finite state automaton.*
- *An arc $((v_1, q_1), (v_2, q_2))$ is only added to G^X if there exists an arc (v_1, v_2) in the original graph G and the arc is labelled with $\sigma \in \Sigma$ so that there exists a transition $q_2 = \delta(q_1, \sigma)$ in the automaton.*
- *The non-negative cost associated with $((v_1, q_1), (v_2, q_2))$ is $l(v_1, v_2)$.*

The resulting graph is not multimodal in the sense that when running the algorithm we do not have to consider what kind of modal changes are made during the journey. There are only allowed modal changes included in the graph.

An algorithm for solving the label constrained shortest path problem in deterministic polynomial time was presented by Barrett et al. (2000). In Holzer (2008) and Barrett et al. (2008) the algorithm was developed further so that the product network does not need to be computed explicitly and the needed space is reduced significantly. In this case the original graph G and the automaton A are given as an input for the algorithm. Then, a normal Dijkstra's algorithm is run from a source vertex s to the target vertex t so that the product graph is calculated only for the arcs and vertices adjacent to the current search space of the Dijkstra's algorithm.

The algorithm 1 presents the methods described above. For the sake of simplicity we assume that all of the arcs are time-dependent, there are no turn restrictions or speed-up techniques applied. Furthermore, the algorithm below allows us to present multiple target vertices but it also works only with a single target.

4.3.3 Speed-up Techniques

Similarly as in the routing in road networks and in public transport networks, there are multiple speed-up techniques developed for the multimodal routing as well (see e.g. Delling et al. 2009). Here, we only briefly present the one exploiting contraction hierarchies for LCSPP-D.

Geisberger et al. (2012b) applied Contraction Hierarchies with LCSPP-D and developed an algorithm called User-Constrained Contraction Hierarchies (UCCH). In the algorithm the contraction is done so that the vertices whose adjacent vertices only belong to the same modal network are contracted. This ensures that all the short cuts travel inside the same modal network and, thus, resulting journey plan obeys L. The algorithm is run in two phases. In the first phase the contraction is performed for each sub-graph that are given as a possible initial or final transport modes. Contraction is stopped when the uncontracted core graph is reach. Then, LCSPP-D can be

run in the resulting graph.

Algorithm 1: Algorithm for solving the label constrained shortest path problem

Data: The original multimodal graph $G = (V, A)$. The source $s \in V$ and the target(s) $t \subseteq V$ of the journey. The finite state automaton $A = (Q, \Sigma, \delta, S, F)$ representing a regular language $L \subseteq \Sigma^*$

Result: A shortest path from s to t which obeys the preference rules stated in L.

PQ $\leftarrow$ a priority queue for the product vertices (v, q) ordered by the non-negative cost from the source vertex to the vertex (v, q) in increasing order;

forall $q_S \in S$ **do**
  PQ.push($(s, q_s), 0$);
end

settled-targets $\leftarrow \emptyset$;

while not *PQ.isempty()* **do**
  $(v, q) \leftarrow$ PQ.pop();
  if $v \in T$ **and** $q \in F$ **then**
    settled-targets $\leftarrow$ settled-targets $\cup\{v\}$;
    if *settled-targets* $= T$ **then**
      stop;
    end
  end
  forall *outgoing arcs* (v, w) **do**
    forall *states* $q' \in \delta(q, label((v, w))$ **do**
      if (w, q') *is a new product vertex* **then**
        PQ.push((w, q'),$dist((s, q_S), (v, q)) + l((v, q), (w, q'))$) ;
      else
        if (w, q') *not yet extracted* **and**
        $dist((s, q_S), (v, q)) + l((v, q), (w, q')) <$
        $dist((s, q_S), (w, q'))$;
        then
          PQ.updatePriority((w, q'),
          $dist((s, q_S), (v, q)) + l((v, q), (w, q'))$);
        end
      end
    end
  end
  stop
end

5 Preference Modelling in Multimodal Routing

As stated earlier our view is that development is needed in routing to improve the modelling of the traveller's preferences. In this section we will propose some methods on how this feature could be improved in multimodal routing. First, we study the methods that could improve the preference modelling in LCSPP-D. Then, we test the improvements in a small sample network.

5.1 Preference Modelling using Label Constrained Shortest Path Problem Dijkstra's Algorithm

The initial goal with the routing algorithm is to find relevant and attractive travelling options, i.e. journey plans, for the traveller given the starting point and the destination. So far, all the algorithms presented above have been optimizing only the travelling time. Thus, the fastest path has been the optimal path. However, it can be easily argued how realistic this model is. Naturally, factors such as congestion, weather, and number of vehicle transfers during the journey also contribute to the attractiveness of a journey plan. Furthermore, individual travellers weight these factors differently. E.g. some travellers always minimize the travelling time while others do not really care about the time as long as they can travel the route without any vehicle transfers.

Because of the reasons stated above also other factors in addition to the travelling time should be taken into account. E.g. Aifadopoulou et al. (2007) proposed a type of a multicriteria optimization solution where they applied linear penalties that depend on the primary optimization variable such as the travel time. However, with the LCSPP-D the problem can be addressed without any modifications on the objective function.

5.1.1 Preference Modelling Using Finite State Automata

Multiple studies show that LCSPP-D can be also exploited in preference modelling (see e.g. Pajor 2009, Dibbelt 2016). With LCSPP-D the regular language restricting the sequences of allowed travelling modes can be designed in the way that it also describes the traveller's preferences. A simple example is shown in the figure 13. Here, the idea is to model journeys performed using only public transport modes with no transfers. The actual structure of the finite state automaton depends off course on the way the multimodal network is constructed and, thus, with different kind of networks functionally similar finite state automata might look completely different.

E.g. in the current example the network would be time-dependent and the arcs linking different modal networks would labelled with a *link* label. Furthermore, arcs leaving the connection specific vertices inside a single stop would be labelled with a $transfer$ label. In the figure, we have a finite state automaton with four states q_0, q_1, q_2, and q_3. States q_0 and q_1 are marked as initial states. Thus, the corresponding journey could start by walking or with one of the public transport modes. The automaton has three final states q_0, q_1, and q_3. The key point here is that the finite state automaton only allows to use the transfer links once. Thus, if public transport modes would be exploited they could be used only once and no vehicle transfers would be allowed.

5.1.2 Multiple Relevant Travelling Options

As stated earlier the goal of the routing algorithm is to return a journey plan given the starting point and the destination. Furthermore, the section above showed that with LCSPP-D traveller's preferences can be taken into account with a proper design of the formal language used in the algorithm. Next, we will present our view of how the preferences should be taken into account in the routing.

There are multiple things that can be encoded in the finite state automaton to describe the traveller's preferences. Problem is that traveller's preferences

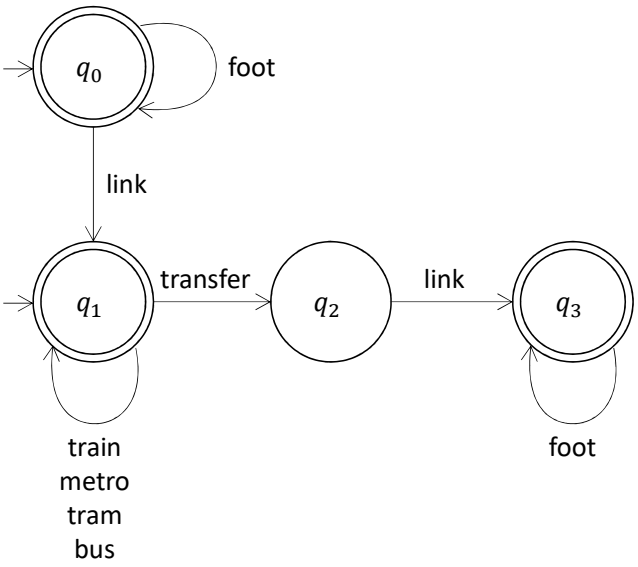


Figure 13: Example of a finite state automaton modelling traveller's preferences.

can depend on temporary variables such as local weather, fatigue or travelling companion. From the routing point of view these kind of temporary variables can be difficult to implement. Furthermore, e.g. Dibbelt (2016) pointed out that the exact modelling of preferences would also require users to set parameters into the application before understanding what are the real effects of these parameters to the returned journey plan. Thus, we would like to avoid modelling of the traveller's preferences too accurately.

This is why we suggest that instead of trying to find the optimal solution we could try to identify a set of possibly optimal journey plans. This means that instead of returning a single journey plan the platform should return a set of attractive plans. Then, the traveller could select the one that is the most suitable for the specific moment.

With the LCSPP-D this can be easily done if there are multiple finite state automata associated with the traveller. Each of the finite state automaton could represent the traveller's preferences in a specific situation. Then, the LCSPP-D could be run parallel so that the actual operator platform could

return a set of relevant travelling options. After this the user could select the plan fitting best to the current situation and start to optimize arrival or departure time.

It could be argued that how important is the preferences modelling if still multiple journey plans will be returned. However, there is a major transition going on in the field of passenger transport. The amount of new service models in this field is constantly growing. Thus, the need for the preference modelling is also growing.

5.1.3 User Segmentation

In the previous section we stated that multiple finite state automata could be used to identify the set of relevant travelling options. Each of these automata should model the traveller's preferences in different situations. A relevant question is that how multiple individual automata can be designed for each of the traveller.

Instead of designing completely new finite state automata for each of the users we could exploit user segmentation. Thus, the travellers should be segmented into similar groups and, then, individual finite state automata should be designed for each of the groups. E.g. Anable (2005) divided the travellers into six clusters based on their attitude statements. Moreover, Stenfors (2017) showed that distinctive groups can be identified using travelling data and constructed mobility archetypes to be used when planning new mobility services. Similar results could be used to identify the correct clusters for each traveller. Then, predefined finite state automata could be associated with every traveller.

The cluster specific automata design it self is left for further research.

5.2 Proof of the Concept

As stated earlier we would like to utilize multiple finite state automata in order to return a set of relevant journey plans for the traveller. In this section we test this in a proof of concept environment. First, we introduce a sample network where the tests will be run. Then, we construct a test framework which allows us to run LCSPP-D algorithm in the sample network. Finally, we run the algorithm using several finite state automata and take a look at the observed results.

5.2.1 Sample Network

We will implement our tests in a small sample network presented in figure 14. The network is highly simplified compared to a real multimodal network but in our literature review we showed that the used algorithms would scale up to more complicated and realistic networks as well. In our sample network we have a single metro line with two stops, two bus lines, and a simple road network. In the road network it is possible to travel by car and by walking.

The network consists of three separate sub-networks. The first two sub-networks correspond to the independent travelling modes, i.e. car and walking modes. These consist of 22 vertices representing the intersections and 8 vertices representing the stops intended for the public transport connections. The networks are presented using two files saved in a comma separated value (CSV) format. In the first file each new line represents a vertex. An example is given in the table 1. Here, we have the vertex name and the corresponding x and y coordinates on each line. Since in our sample network the car and walking modes exploit the same network this file can be actually shared among the corresponding two sub-networks.

The second file describes the arcs connecting the vertices. Here, each line correspond to an arc in the sub-network. An example is given in the table 2. On each line we have the source and target vertex names followed by the corresponding arc length, i.e. the travelling time. Since travelling by a car is faster than walking separate files are needed for both sub-networks.

Line	Row content
1	1,500,5100
2	2,3200,5100
3	3,500,4400
⋮	⋮

Table 1: An example of the first CSV file describing the sub-network for the independent modes. In the file each line correspond to a vertex in the sub-network. First number is the vertex number followed by the x and y coordinate of the vertex.

Line	Row content
1	1,3,0.0259
2	3,1,0.0259
3	3,4,0.0296
⋮	⋮

Table 2: An example of the second CSV file describing the sub-network for the independent modes. In the file each line correspond to an arc in the sub-network. On the lines we have the names of the source and target vertices follwed by the travelling times between these vertices.

The third sub-network corresponds to the public transport modes, i.e. bus and metro modes. Also here two CSV files are used to desccribe the sub-network. The first file is equal to the first file in the sub-networks presented before. It consists of the vertex names and their coordinates. However, the second file differs since the network is time-dependent and there is a travelling time function associated with each arc. An example of this file is presented in the table 3. Here each line represents a vehicle leaving from a specific stop. Thus, on the line we have the names of the source and target vertices, the label of the corresponding travelling mode, and the departure and travelling time.

All networks were built specifically for these tests. Comparison was made how the actual road and public transport networks are typically presented so that the constructed test framework could utilize actual road networks as well in the future.

Line	Row content
1	B1-1,B1-2,bus,0.3333,0.0389
2	B1-1,B1-2,bus,0.3403,0.0389
3	B1-1,B1-2,bus,0.3472,0.0389
⋮	⋮

Table 3: An example of the second CSV file describing the sub-network for the public transport modes. In the file each line correspond to a vehicle leaving from a specific stop. On the line we have the names of the source and target vertices, label of the corresponding travelling mode, and departure and travelling times.

5.2.2 Implemented Test Framework

Next, we need a framework where we can run the tests and utilize the sample network described above. The main components needed in this framework were constructed using object oriented programming and the actual tests were run using simple scripts. Everything was implemented using the R language (R Development Core Team 2008). In total, we constructed three modular class structures to be used in all tests and a single function to run the algorithm. Then, we had multiple scripts in place to run the tests with different parameters. Furthermore, separate scripts were needed to visualize the obtained results. These were implemented using the R language's *ggplot* package (Wickham 2009). For simplicity we present here only the designed class structures and the main function used to run the LCSPP-D algorithm.

First, we present the class structure that allows the optimization algorithm to query information about the used network. This can be found from the figure 15. All constructed class structures are presented using the Unified Modelling Language (UML) type of notation. Each container in the figures represents a class. The class name is presented on the top. Class attributes are presented below the name and the class methods in the bottom part of the container. After each attribute or method there is also a brief description available. Furthermore, there are two type of classes in the figure, base classes and derived classes. A base class stands on its own and all the attributes and methods available are described inside its own container. A derived class represents a specialized version of the base class. It inherits all the

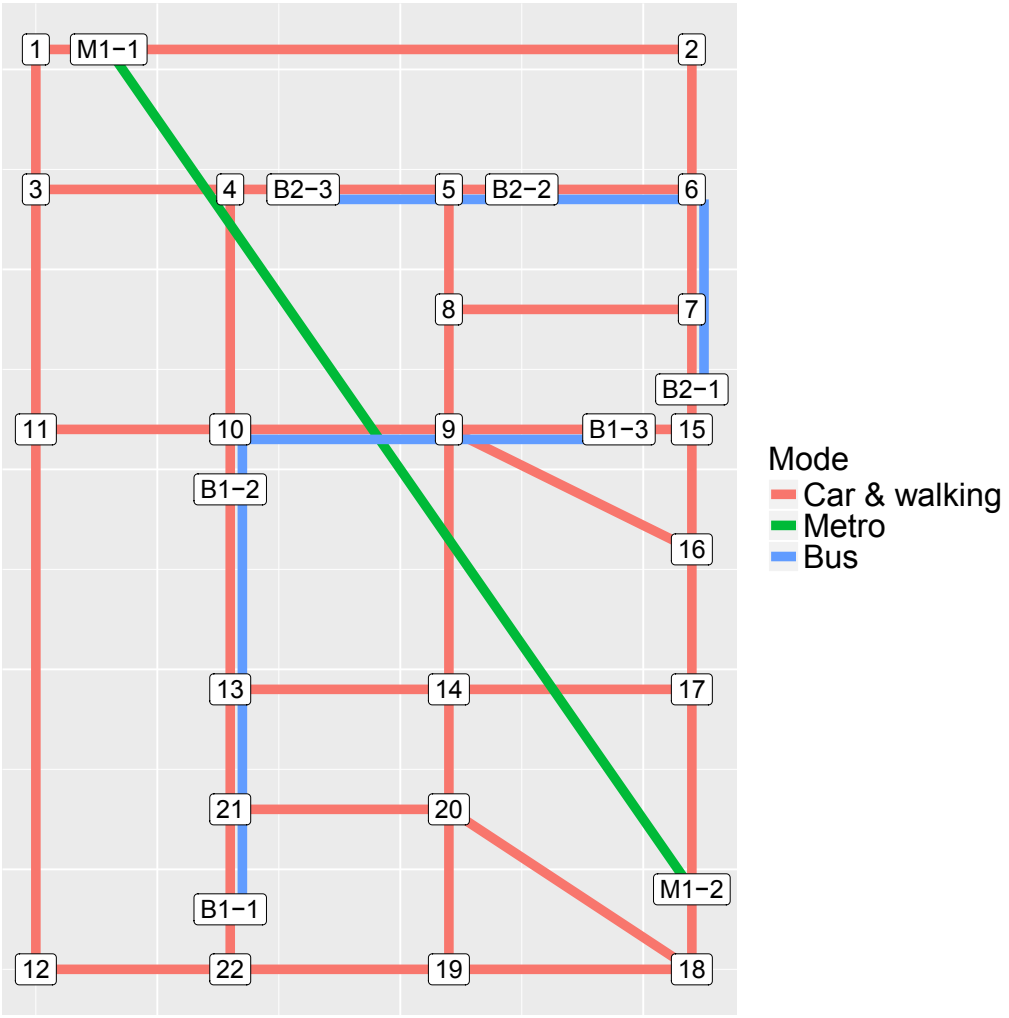


Figure 14: The sample network where all the tests were executed. There is a single metro line and two bus lines in the network. In the network it is possible to travel using independent modes, i.e. car or walking modes, or using the public transport modes.

components of the base class but can also overwrite specific functionalities and even present new ones.

From the figure 15 we can see that we have two base classes, *Network* and *SubNetwork* classes. *Network* class represents the complete network and the *SubNetwork* class a component of this where only a single mode can be exploited. Furthermore, we have a derived class called *SubNetorkTimeDependent*.

It has the same functionalities as its base class, $SubNetwork$, but the travel times are time dependent and calculated using a travel time function associated with the corresponding arc.

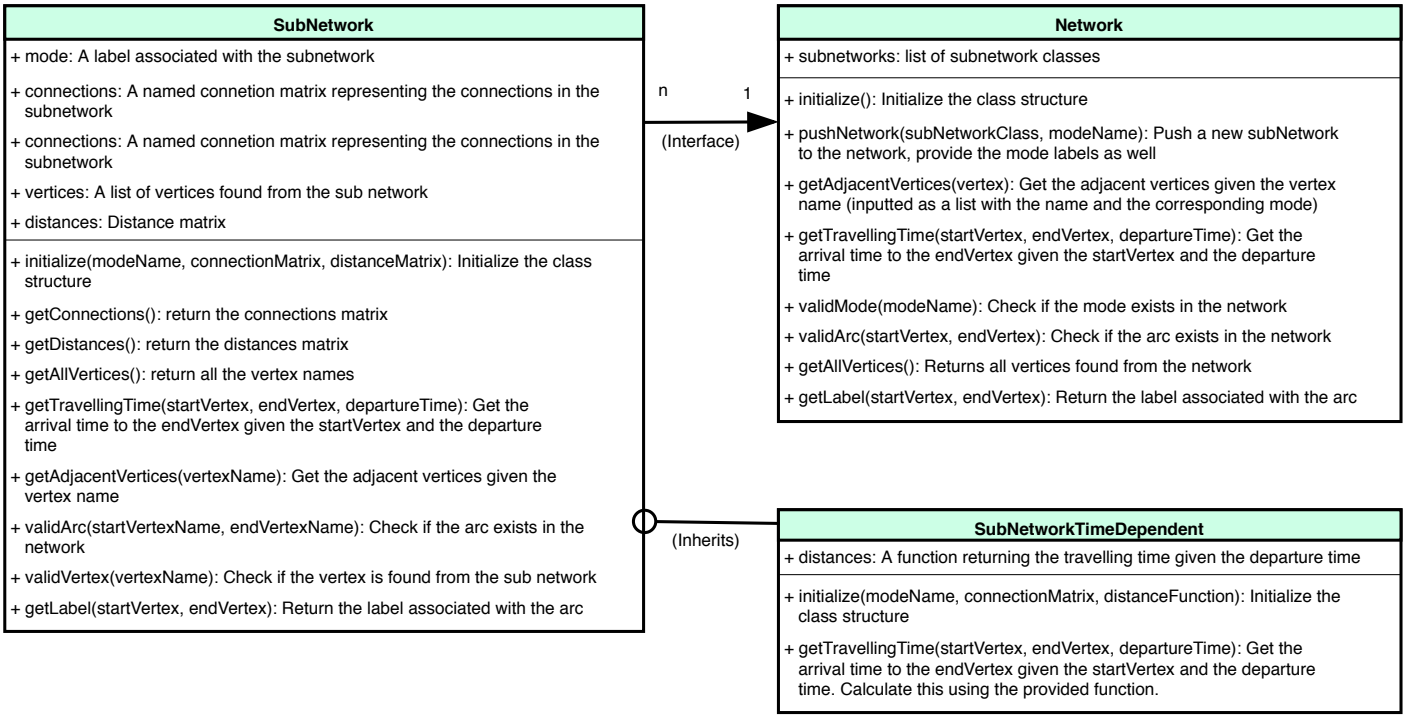


Figure 15: Class diagram of the classes used to model the networks inside the optimization algorithm. The diagram consists of two base classes, $Network$ and $SubNetwork$, and one derived class, $SubNetworkTimeDependent$. Furthermore, there is an interface between the classes $Network$ and $SubNetwork$. It means that there is always one or more $SubNetwork$ objects below a single $Network$ object and this $Network$ can access these $SubNetwork$ objects and their methods.

In addition, we have a single interface between two classes in the figure 15. Interface here means that the $Network$ class can reach all the $SubNetwork$ classes and utilize their methods inside its own methods. It can e.g. query for a travelling time between a two specific vertices from the corresponding $SubNetwork$ class. This way the implementation is modular and it is easier to e.g. add as many $SubNetworks$, i.e. travelling modes, to the set up as needed.

Overall, the class structure constructed to model the network is designed to

be easily usable by the LCSPP-D algorithm and all the methods used in the algorithm can be found from the figure 15. The modular design also means that it should be possible to model any kind of transportation network with this class structure.

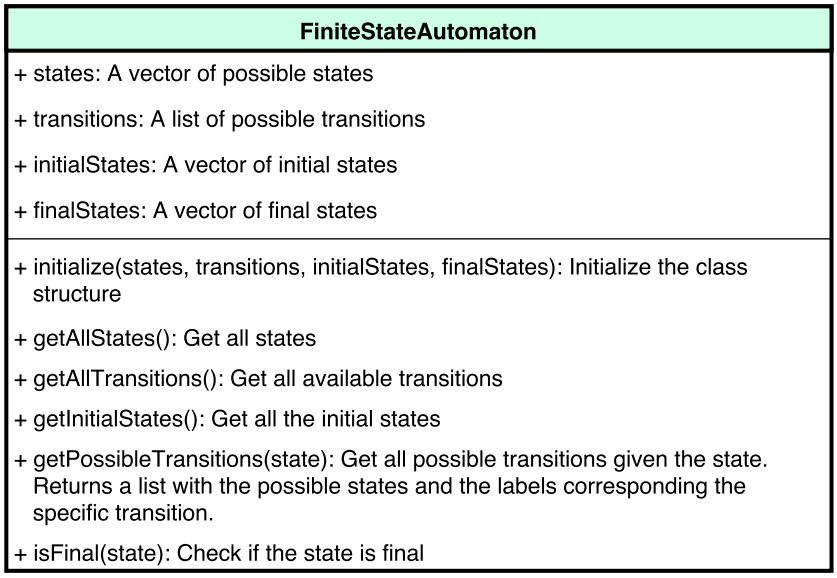


Figure 16: Class modelling the finite state automaton used in the LCSPP-D algorithm. The class attributes and methods are presented below the class name in the figure, respectively. Also brief descriptions of all the functionalities is available.

The second class structure needed in the framework is presented in figure 16. It represents the finite state automaton needed in the LCSPP-D algorithm. A more formal definition of a finite state automaton is presented in the section 4.3.2. Also here the class structure is designed to be modular and easily usable in the optimization algorithm. All the methods needed in the LCSPP-D algorithm can be found from the figure 16.

The third class structure needed in the LCSPP-D algorithm is the priority queue. This is presented in the figure 17. This is a generic implementation of a priority queue. Any kind of tuples associated with a priority can be placed in the queue. Then, the priority queue sorts the tuples accordingly. With the LCSPP-D algorithm it means that product vertices, i.e. tuples consisting of a vertex and a state, are placed in the queue associated with the travel time

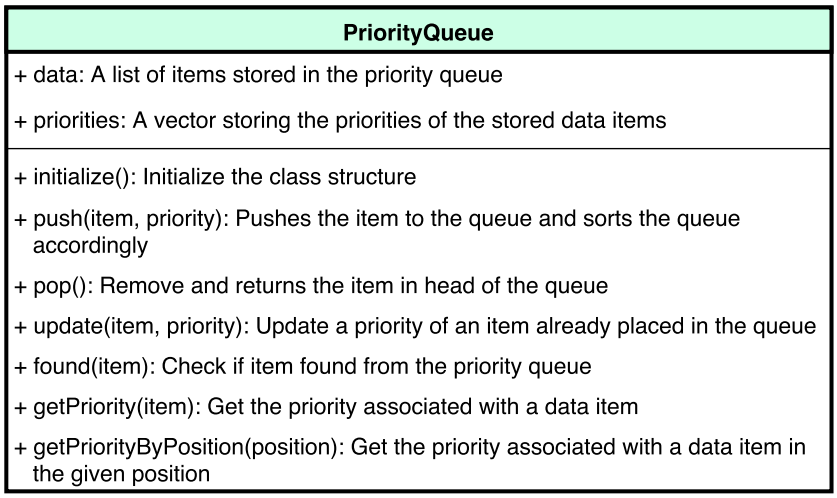


Figure 17: The *PrirityQueue* class used in the LCSPP-D algorithm. The class attributes and methods are presented below the class name in the figure, respectively. Also brief descriptions of all the functionalities is available.

as the priority.

Now we have all the necessary class structures defined for tests. Next, we will shortly describe the function running the LCSPP-D algorithm. Also here for simplicity we only give a brief description of the function. The actual source code is presented in the appendix A.

The function is called $LCSPPDijkstra$ and it takes five arguments as an input:

- **Network class:** This class describes the complete network where the algorithm is run. All the $SubNetwork$ classes are also found under the $Network$ class. Separate scripts were needed to transform the files described in the section above to corresponding classes.
- **FiniteStateAutomaton class:** This class describes the finite state automaton restricting the sequence of used travelling modes in the algorithm. Separate scripts were in place to define all needed $FiniteStateAutomaton$ classes. The actual used finite state automata are described in the next section.
- **Source vertices:** A vector consisting of the source vertex names

- **Target vertices:** A vector consisting of the target vertex names
- **Departure time:** The time of the departure. Since part of the used network is time-dependent and the travel time depends on the actual time of the day the departure time is also needed.

When the function execution starts it first initializes the *PriorityQueue* class used in the algorithm. Then, it iterates like described earlier in the algorithm 1 found from the section 4.3.2. A technological perspective can be found from the source code presented in the appendix A.

After the run is completed the function returns a list consisting of all the necessary things for us to analyse the results. The return consists of the following objects:

- **Targets:** A vector of target vertex names. Originally this is an input of the function.
- **Arrival times:** Arrival times to the target vertices in the corresponding order as the provided targets.
- **Final states:** Final states in the target vertices in the corresponding order as the the provided targets.
- **Previous vertices:** A named vector of all vertices found from the complete network. Under each name there is the corresponding previous vertex determined by the LCSPP-D algorithm. If the algorithm did not visit a specific vertex NA is introduced here. This vector allows us to determine the actual path the algorithm has travelled to the target vertex.
- **Sources:** A vector of source vertex names. Originally this is an input of the function.

Using these outputs and the original files describing the network, we can visualize the results. This is done next. Also the used parameters such as the finite state automata are described more closely here.

5.2.3 Test Runs

The sample network was presented in figure 14. We decided to run our tests so that in all cases we optimize the route between the vertex 19 and the vertex 4. First, we wanted to get a control result where no restrictions on the sequence of used travelling modes were made. The finite state automata corresponding to this situation is presented in figure 22(a). The resulting journey plan is also presented in 18. We can see that the resulting route starts from the vertex 19 with a car to the $M1-2$ metro station. Then, it continues to the $M1-1$ using the metro and, finally, travels to the destination using again a car. This is the fastest route between the two vertices but it can be argued how realistic it is to have a car at use at the second metro station and, furthermore, does this route model the traveller's preferences in any way. Thus, we need to make some improvements to the used finite state automata.

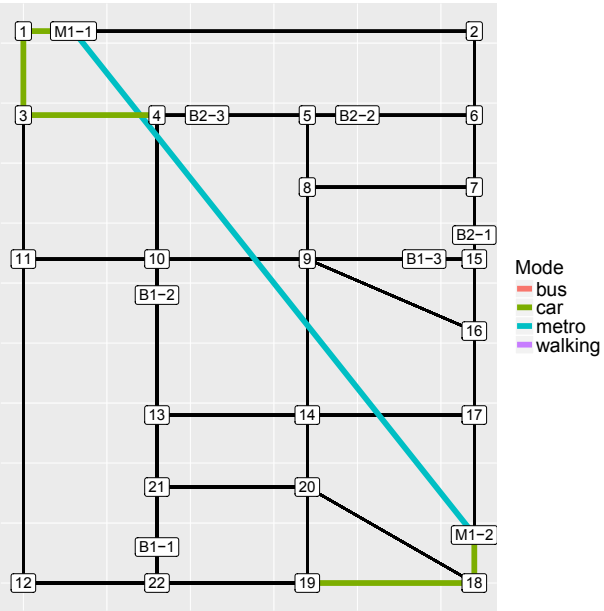


Figure 18: A journey plan to travel from the vertex 19 to the vertex 4 calculated using the finite state automaton presented in figure 22(a).

Next, we would like to return multiple journey plans and incorporate some more advanced finite state automata into the set up. It has to be noted that

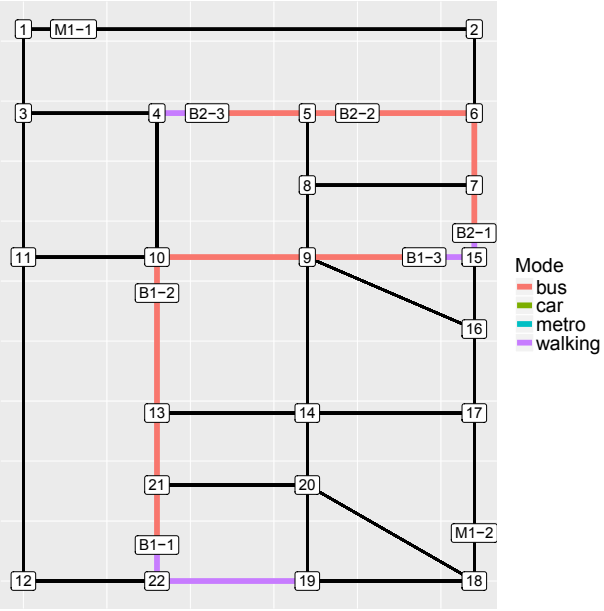


Figure 19: A journey plan to travel from the vertex 19 to the vertex 4 calculated using the finite state automaton presented in figure 22(b).

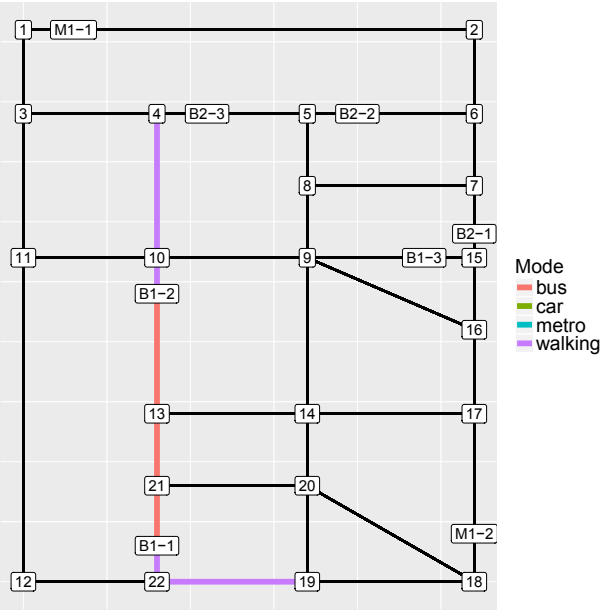


Figure 20: A journey plan to travel from the vertex 19 to the vertex 4 calculated using the finite state automaton presented in figure 22(c).

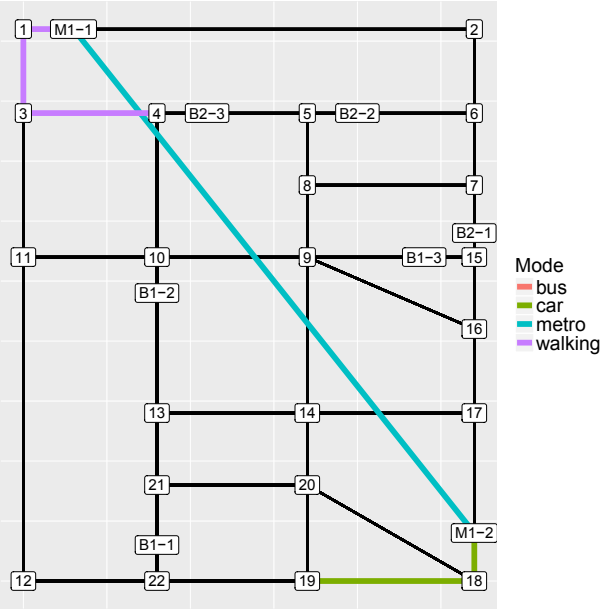
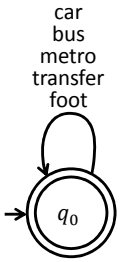


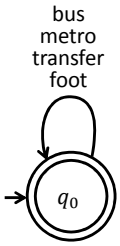
Figure 21: A journey plan to travel from the vertex 19 to the vertex 4 calculated using the finite state automaton presented in figure 22(d).

the used automata are just illustrative and implemented to demonstrate how the preferences could be modelled with finite state automata. We have three different kind of automata. These are presented in figures 22(b)-22(d). First, we have the automaton which states that only public transport can be used but does not impose any other restrictions to the resulting journey plan. Secondly, we have an automaton which states that only public transport can be used and no vehicle transfers are allowed. Lastly, we have an automaton which tries to demonstrate that also more exact modelling can be done with the finite state automata. Here, we say that the traveller wants to find the optimal path which starts by taking a car to a metro station and then continuing using only public transport options.

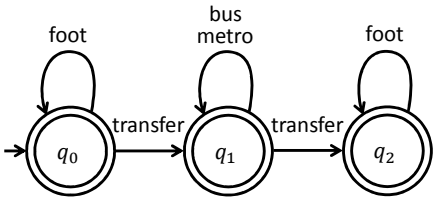
Using the three automata described above we will now run the algorithm to optimize the paths between the same vertices as earlier. The resulting journey plans are presented in 19-21. Each of the finite state automata seem to work like we expected. In the figure 19 we have the route using only public transport options. We can see that the resulting journey plan differs a lot from the control result presented in the figure 18. Instead of using the metro,



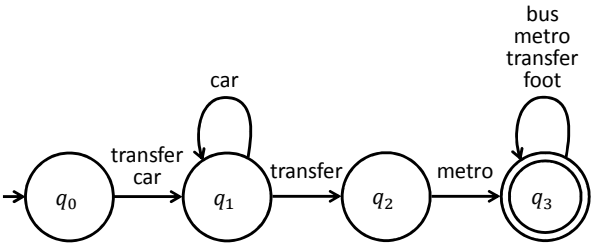
(a) No restrictions



(b) Only public transport



(c) Public transport and only single vehicle transfer



(d) Park-and-drive in the nearby metro station

Figure 22: Four different finite state automata used to retrieve journey plans presented in figures 18-21.

the traveller now utilizes the bus lines available. Since the car mode is not available this time, it is faster to take the two bus lines and minimize the amount of walking during the journey.

In the figure 20 we have the route returned using the second automaton. The restriction here was to use public transport options but only allow a single vehicle transfers. Thus, the resulting journey plan removes the last bus leg seen in the previous version.

Lastly, in the figure 21 we have the journey plan returned using the third automaton. This begins as the control result presented earlier but ends by walking since the traveller has no car available at the second metro station.

5.2.4 Results and Further Improvements

In this section we wanted to study if LCSPP-D algorithm and multiple finite state automata could be used to retrieve a set of journey plans from a single network. By looking at the results calculated above we can say that this is definitely possible. We applied four different finite state automata and the returned journey plans all obey the preference rules stated in the automata.

Our literature review showed also that the performance of the used LCSPP-D algorithm can be boosted using applicable speed-up techniques. This means that the results should also be scalable to a larger and more realistic network as well. The presented test framework is modular and build so that any network can be input as long as it is stored in a suitable format. However, since there is a major overhead in building a new network using actual road and public transport network, this is left for further research. Naturally, also some modifications to the framework will be needed to include the speed-up techniques into the calculations.

Earlier we stated also that a question left for further research is how well the finite state automata can actually model the traveller's preferences. Furthermore, our proposition was to cluster the travellers into groups with similar preferences and this way the finite state automata modelling the travellers'

preferences could be shared within the groups. How the clustering should be done and how the cluster specific finite state automata should be designed is something that should be studied more closely in the future as well.

6 Discussion and conclusions

In this study, we looked at the ongoing change in the field of passenger transport. First, we studied this from the travellers' perspective. We saw new transport service providers such as shared rides and cars entering the market and new service models such as Mobility as a Service (MaaS) operators evolving. Overall, in the upcoming years there will be more variety in the available travelling options and accessibility of these options will be improved significantly for the traveller.

We also stated that there is a need for a new technology platform integrating the transport service providers to MaaS operators for information exchange and to matchmake travel demand and supply. We presented a reference architecture for the platform and estimated the need for development in the different components of the platform. We identified that development is at least needed to improve the modelling of the travellers' preferences in multimodal routing algorithms.

Then, we conducted a literature review to identify routing algorithms capable of modelling the traveller's preferences. An algorithm called label constrained shortest path problem Dijkstra's (LCSPP-D) algorithm is one typically used in this context. We proposed two ways to improve the preference modelling with the LCSPP-D algorithm.

Firstly, the travellers should be clustered into similar groups so that the parameters describing the preferences could be shared within the group. This way more emphasis could be given to the optimization of the group specific parameters. Secondly, instead of returning journey plans using a single objective function, a set of journey plans should be returned where each would describe the travellers' preferences in different situations. Then, depending

on temporary variables such as the weather, travelling companion or the amount of luggage the traveller could select the plan most suitable for the specific situation.

We decided to evaluate the second improvement more closely. LCSPP-D algorithm uses regular languages to restrict the sequences of the used travelling modes. By providing several regular languages for the algorithm it is possible to return different kind of journey plans for a single trip. Furthermore, with a proper design of the associated regular languages each of the returned journey plans could model the traveller's preferences in different situations.

Lastly, we implemented a modular test framework where we could run the LCSPP-D algorithm in a sample network. We designed four regular languages to describe the travellers' preferences and used these to return journey plans from the sample network. The results show that journey plans modelling the travellers' preferences can be returned and for a single trip we can return multiple plans each describing different kind of preferences.

The aim for this study was to identify multimodal routing algorithms capable of modelling the travellers' preferences and find ways to improve this property of the algorithms. We proposed two ways to improve the preference modelling with the LCSPP-D algorithm. Furthermore, it should be possible to apply similar improvements with other multimodal routing algorithms as well. However, further research is needed to study how well the algorithm can actually model the traveller's preferences and how the regular languages used in the algorithm should be defined. Also, further research is needed to study how the clustering of the travellers should be done. Overall, based on this study we can say that more development and new technological solutions will be needed to support the change currently happening in the field of passenger transport.

References

Georgia Aifadopoulou, Athanasios Ziliaskopoulos, and Evangelia Chrisohoou. Multiobjective optimum path algorithm for passenger pretrip planning in multimodal transportation networks. *Transportation Research Record: Journal of the Transportation Research Board*, (2032):26–34, 2007.

Jillian Anable. 'complacent car addicts' or 'aspiring environmentalists'? identifying travel behaviour segments using attitude theory. *Transport Policy*, 12(1):65–78, 2005.

Chris Barrett, Riko Jacob, and Madhav Marathe. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.

Chris Barrett, Keith Bisset, Martin Holzer, Goran Konjevod, Madhav Marathe, and Dorothea Wagner. Engineering label-constrained shortest-path algorithms. *Algorithmic aspects in information and management*, pages 27–37, 2008.

Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. In *Algorithm Engineering*, pages 19–80. Springer, 2016.

Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *Experimental Algorithms*, pages 303–318, 2008.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

Gerth Stølting Brodal and Riko Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electronic Notes in Theoretical Computer Science*, 92:3–15, 2004.

Tom Caldwell. On finding minimum routes in a network with turn penalties. *Communications of the ACM*, 4(2):107–108, 1961.

George Dantzig. *Linear programming and extensions*. Princeton university press, 2016.

Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating multi-modal route planning by access-nodes. In *ESA*, volume 5757, pages 587–598. Springer, 2009.

Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Customizable route planning. In *International Symposium on Experimental Algorithms*, pages 376–387. Springer, 2011a.

Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Customizable route planning. In *International Symposium on Experimental Algorithms*, pages 376–387. Springer, 2011b.

Julian Matthias Dibbelt. *Engineering Algorithms for Route Planning in Multimodal Transportation Networks*. PhD thesis, Karlsruhe, Karlsruher Institut für Technologie (KIT), Diss., 2016, 2016.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Yann Disser, Matthias Muller-Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. *Lecture Notes in Computer Science*, 5038:347–362, 2008.

Yvan Dumas, Jacques Desrosiers, and Francois Soumis. The pickup and delivery problem with time windows. *European journal of operational research*, 54(1):7–22, 1991.

David Eppstein and Michael T Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 16. ACM, 2008.

Robert Geisberger. Contraction of timetable networks with realistic transfers. In *SEA*, pages 71–82. Springer, 2010.

Robert Geisberger and Christian Vetter. Efficient routing in road networks with turn costs. *SEA*, 11:100–111, 2011.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. *Experimental Algorithms*, pages 319–333, 2008.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012a.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012b.

Andrew V Goldberg and Renato Fonseca F Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, pages 26–40, 2005.

Taina Haapamäki and Sami Mäkinen. Open mobility markets. *1st International Conference on Mobility as a Service, Tampere 28.-29.11.2017*, 2017.

Martin Holzer. *Engineering planar-separator and shortest-path algorithms.* PhD thesis, Karlsruhe Institute of Technology, 2008.

Jeremy Horpedahl. Ideology uber alles? Economics bloggers on Uber, Lyft, and other transportation network companies. *Econ Journal Watch*, 12(3), 2015.

Brigitte Jaumard, Frederic Semet, and Tsevi Vovor. A generalized linear programming model for nurse scheduling. *European journal of operational research*, 107(1):1–18, 1998.

Vachaspathi P Kompella, Joseph C Pasquale, and George C Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking (TON)*, 1(3):286–292, 1993.

Feng Li and Jason Whalley. Deconstruction of the telecommunications industry: from value chains to value networks. *Telecommunications Policy*, 26(9):451 – 472, 2002.

Matthias Müller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Timetable information: Models and algorithms. In *Algorithmic Methods for Railway Optimization*, pages 67–90. Springer, 2007.

Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)*, 37(3):607–625, 1990.

Thomas Pajor. Multi-modal route planning. *Universität Karlsruhe*, 2009.

Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *Journal of Experimental Algorithmics (JEA)*, 12:2–4, 2008.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. URL http://www.R-project.org.

Aleksi Stenfors. Identifying mobility user segments based on everyday travel data. *Aalto University*, 2017.

Dirck Van Vliet. Improved shortest path algorithms for transport networks. *Transportation Research*, 12(1):7–20, 1978.

Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. URL http://ggplot2.org.

A Label Constrained Shortest Path Problem with R

```
#Given the network, finite state automata, sources and targets,
#find the fastest path between the sources and targets which obeys
#the sequence of travelling modes defined in the finite state automata
LCSPPDijkstra = function(network, finite_state_machine, sources, targets,
                         start_time = 0){
  #initialize the priority queue
  priority_queue <- PriorityQueue$new()

  #initialize the vector to save the previous vertices,
  #with this this correct path is derived afterwards
  previous_vertices <- character(0)

  #push all sources to the priority queue
  invisible(lapply(sources, function(x) {
    priority_queue$push(
      list(
        vertex=x,
        state=finite_state_machine$getInitialState()),
      start_time)
    }))

  #push all the sources in the used_product_vertices list
  used_product_vertices <- lapply(sources, function(x) {
    list(vertex=x, state=finite_state_machine$getInitialState())
    })

  #initialize the settled targets
  settled_targets <- list()
  settled_targets_time <- numeric(0)
  settled_targets_state <- character(0)

  #iterate until the shortest path is found
  while(priority_queue$size() > 0){
    #pop the first vertex state combination from the priority queue
    extracted_time <- priority_queue$getPriorityByPos()
    extracted_product_vertex <- priority_queue$pop()

    #if possible add extracted_product_vertex$vertex to the
    #settled_targets
    if(containsVertex(targets, extracted_product_vertex$vertex) &&
       finite_state_machine$isFinal(extracted_product_vertex$state)){

      settled_targets[[length(settled_targets)+1]] <-
        extracted_product_vertex$vertex
      settled_targets_time <- c(settled_targets_time,
                                extracted_time)
      settled_targets_state <- c(settled_targets_state,
                                 extracted_product_vertex$state)

      #if all targets are settled stop the shile loop
      if(identical(settled_targets, targets)){
        break
      }
    }

    #get the outgoing arcs from extracted_product_vertex$vertex
    adjacent_vertices <-
      network$getAdjacents(extracted_product_vertex$vertex)
```

```
#loop over these
for(i in adjacent_vertices){
  #get the possible transitions
  possible_transitions <-
    finite_state_machine$getPossibleTransitions(
      extracted_product_vertex$state)

  #loop over these
  for(j in possible_transitions){
    #check if the move is valid
    if(j$label ==
       network$getLabel(extracted_product_vertex$vertex, i)){
      temp_product_vertex <- list(vertex = i, state = j$state)

      #get the distance to the temp vertex
      arrival_time <-
        network$getTravellingTime(extracted_product_vertex$vertex,
                                  temp_product_vertex$vertex,
                                  extracted_time)

      #if the new product vertex not found from
      #the used product vertices
      if(!containsProductVertex(used_product_vertices,
                                temp_product_vertex)){
        priority_queue$push(temp_product_vertex, arrival_time)
        used_product_vertices[[length(used_product_vertices)+1]] <-
          temp_product_vertex
        #also update the previous vertex
        previous_vertices <- updatePrevious(
          previous_vertices,
          temp_product_vertex$vertex$name %&%
            "+" %&%
            temp_product_vertex$vertex$mode %&%
            "+" %&%
            temp_product_vertex$state,
          extracted_product_vertex$vertex$name %&%
            "+" %&%
            extracted_product_vertex$vertex$mode %&%
            "+" %&%
            extracted_product_vertex$state
          )

      }
      #update if improved
      else if(priority_queue$found(temp_product_vertex)){
        if(priority_queue$size() > 0 &&
           priority_queue$getPriority(temp_product_vertex) >
           arrival_time){
          priority_queue$update(temp_product_vertex,
                                arrival_time)
          #also update the previous vertex
          previous_vertices <-
            updatePrevious(previous_vertices,
                           temp_product_vertex$vertex$name %&%
                             "+" %&%
                             temp_product_vertex$vertex$mode %&%
                             "+" %&%
                             temp_product_vertex$state,
                           extracted_product_vertex$vertex$name %&%
                             "+" %&%
                             extracted_product_vertex$vertex$mode %&%
                             "+" %&%
                             extracted_product_vertex$state)
```

```
            }
          }
          break
        }
      }
    }

  }
  #collect the wanted results and return them
  return(list(
    targets = settled_targets, #all targets
    times = settled_targets_time, #travelling times to the targets
    states = settled_targets_state, #final states in the targets
    "previous vertices" = previous_vertices, #character vector of
    #previous vertices
    sources = sources #sources
  ))
}
```